Chapter 3

How to add BLE functionality easily using STM32CubeMX

life.augmented

How to add BLE functionality easily using STM32CubeMX

Mission

- Refresh the BLE main basic knowledge and principles
- How to add the STM32_WPAN BLE middleware to an existing project
- Understand the STM32_WPAN middleware architecture basics

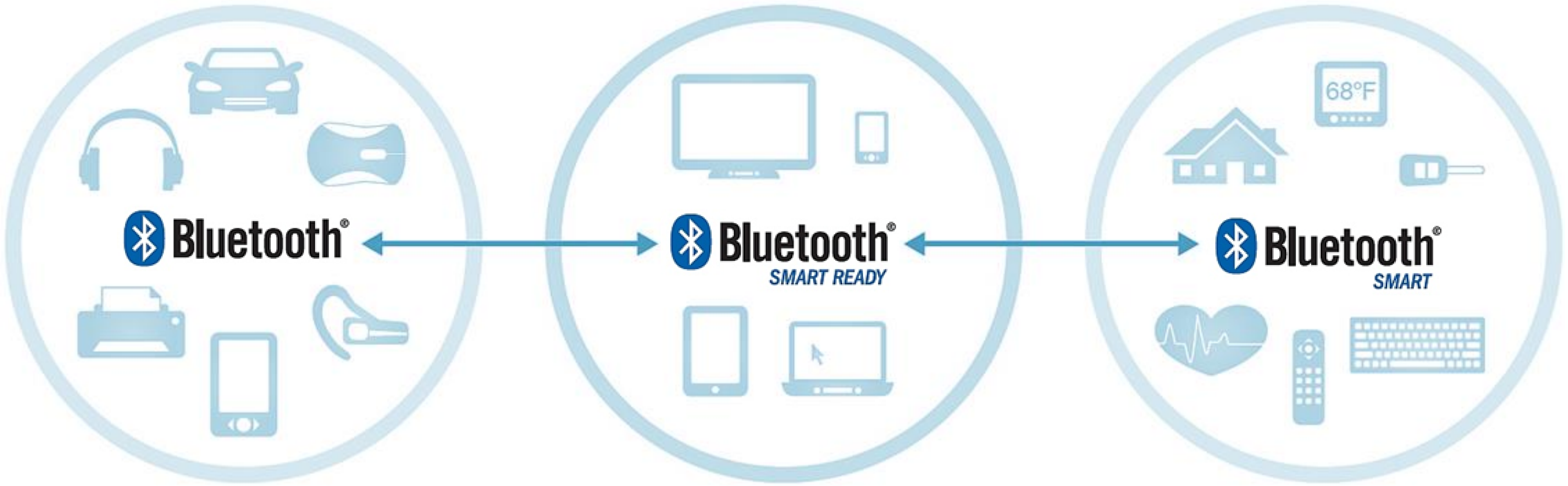
theory | practice

Bluetooth in "15min" ☺

- Target applications
- Key aspects & features
- Basic principles
- Terminology

Bluetooth Smart introduction

Target applications



Wireless devices streaming rich content

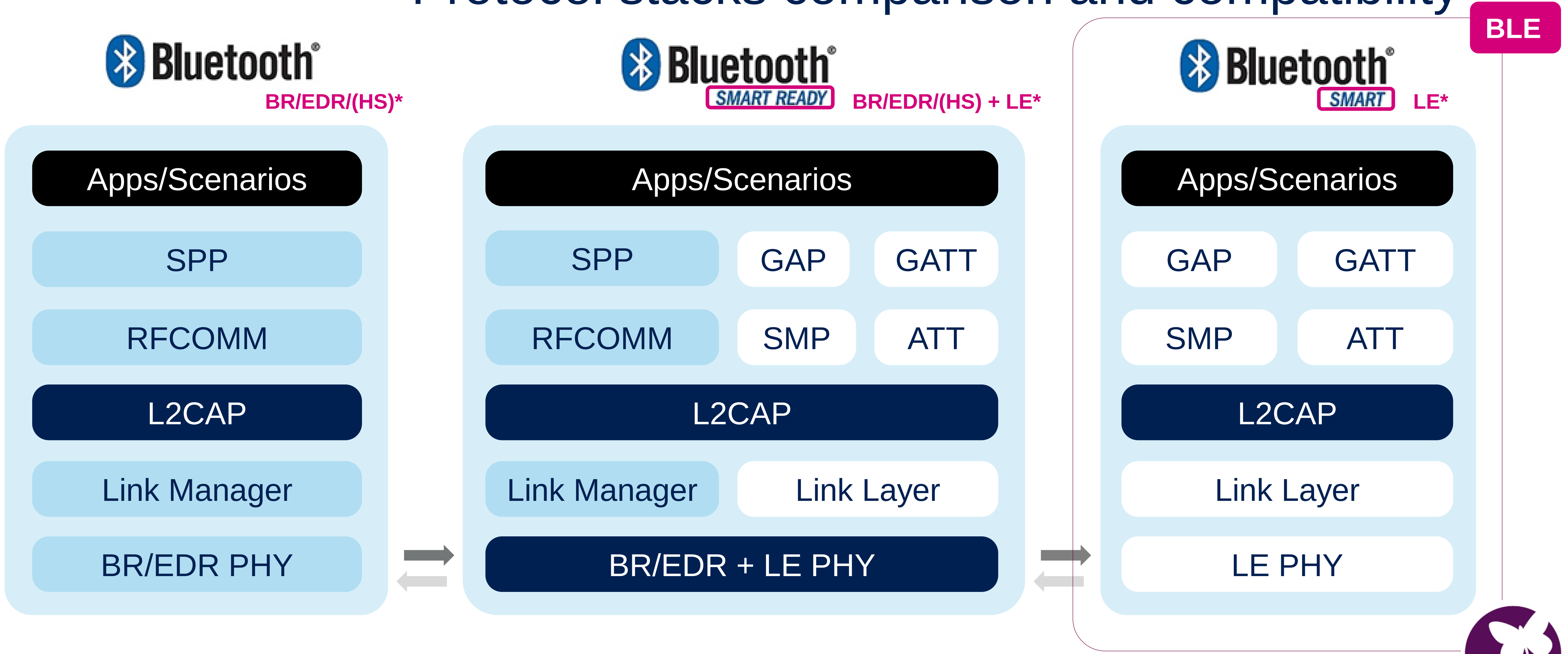
Devices that can connect with both Bluetooth Classic and Bluetooth Smart

Sensor devices, accessories with low data-rate, but high battery lifetime requirements

"SMART READY" and ***"SMART"*** are abandoned markings

Bluetooth Smart introduction

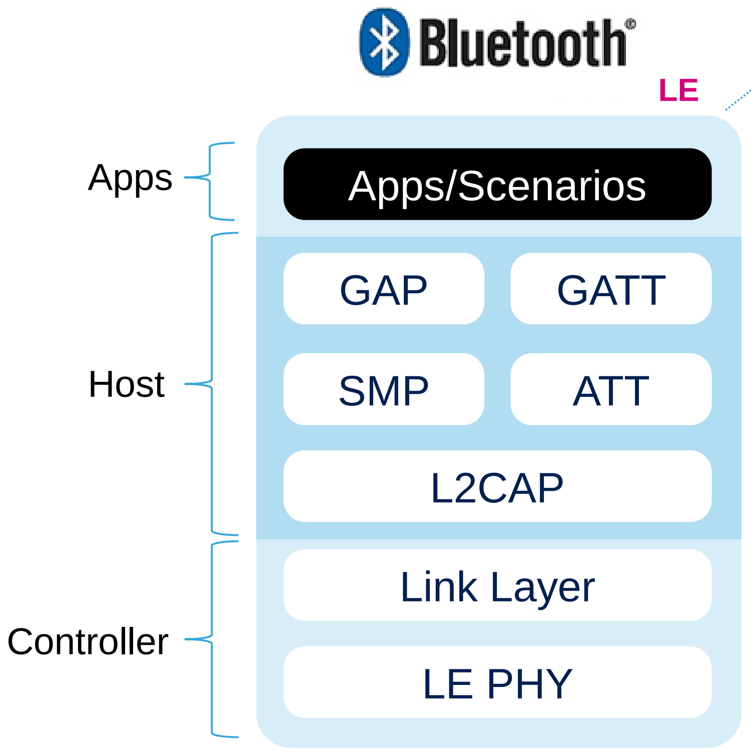
Protocol stacks comparison and compatibility

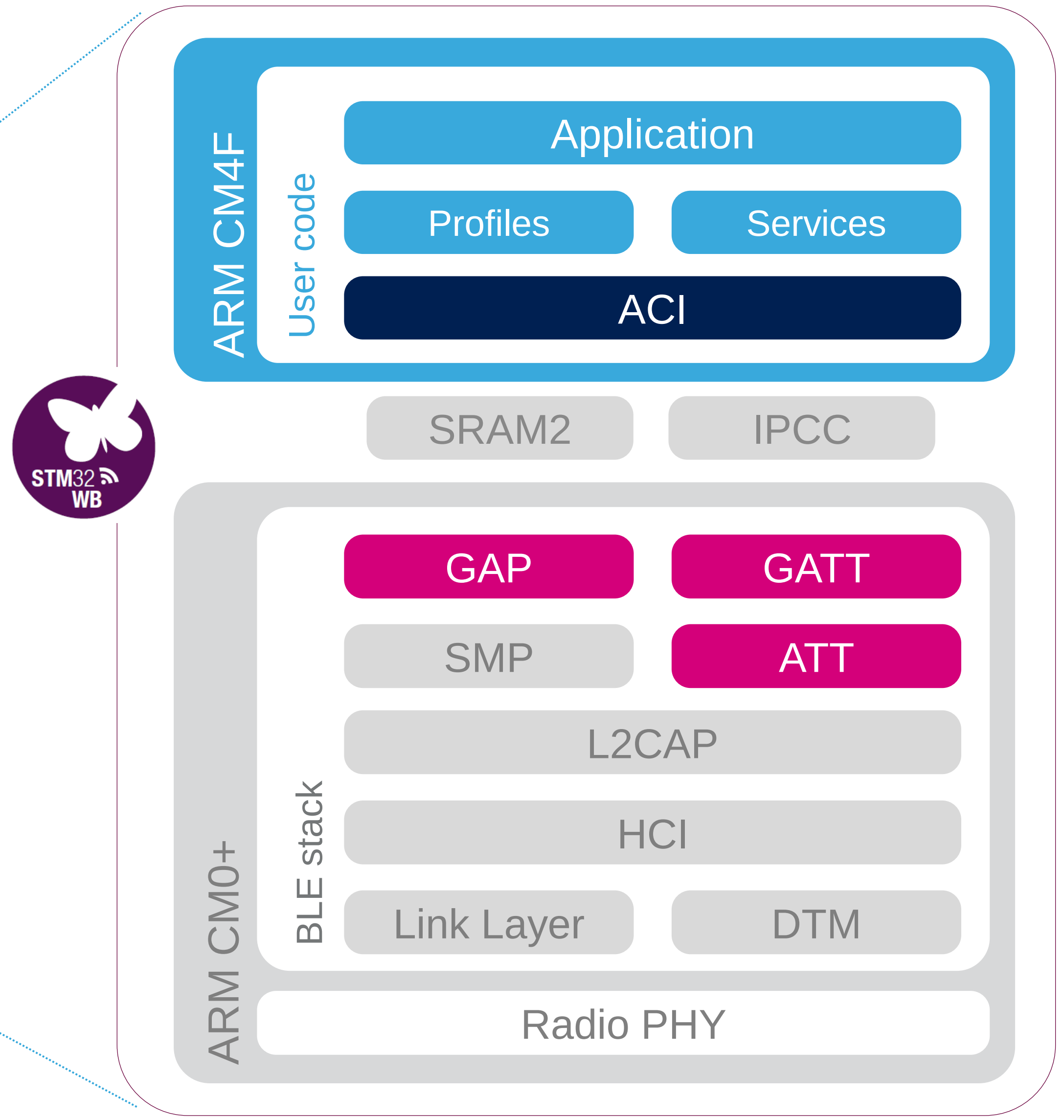


Bluetooth 5 since end of 2016 reflecting Bluetooth core specification v5.0

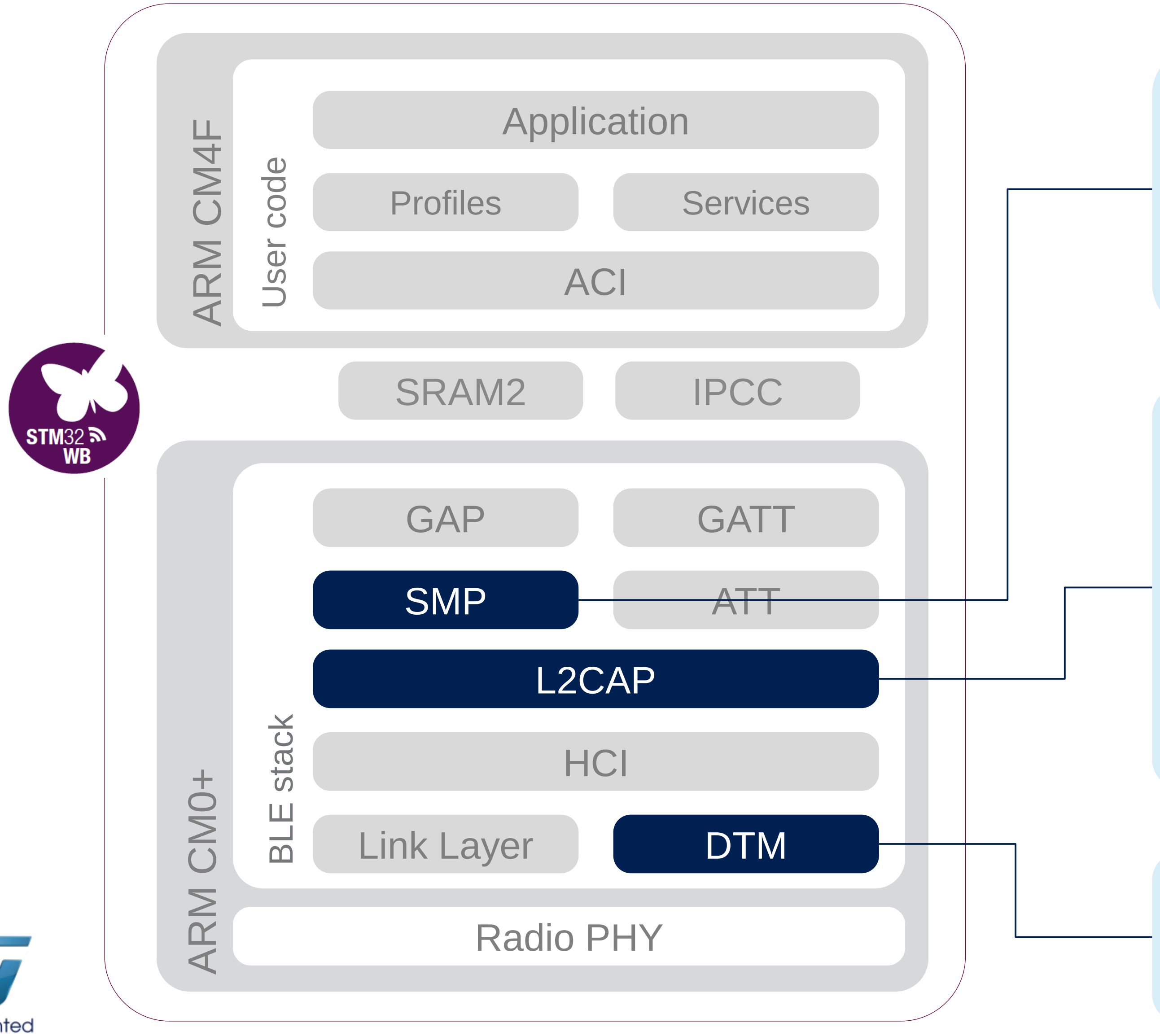
Bluetooth is a **collection of different protocols** grouped together under a single specification

What does it mean for me?





Bluetooth LE various stack layers



SMP – Security Manager Protocol

Responsible for pairing, encryption (AES-128), keys management...

L2CAP – Logical Link Control and Adaptation Protocol

Takes multiple protocols from upper layers and encapsulates them into BLE packets, which are passed to HCI

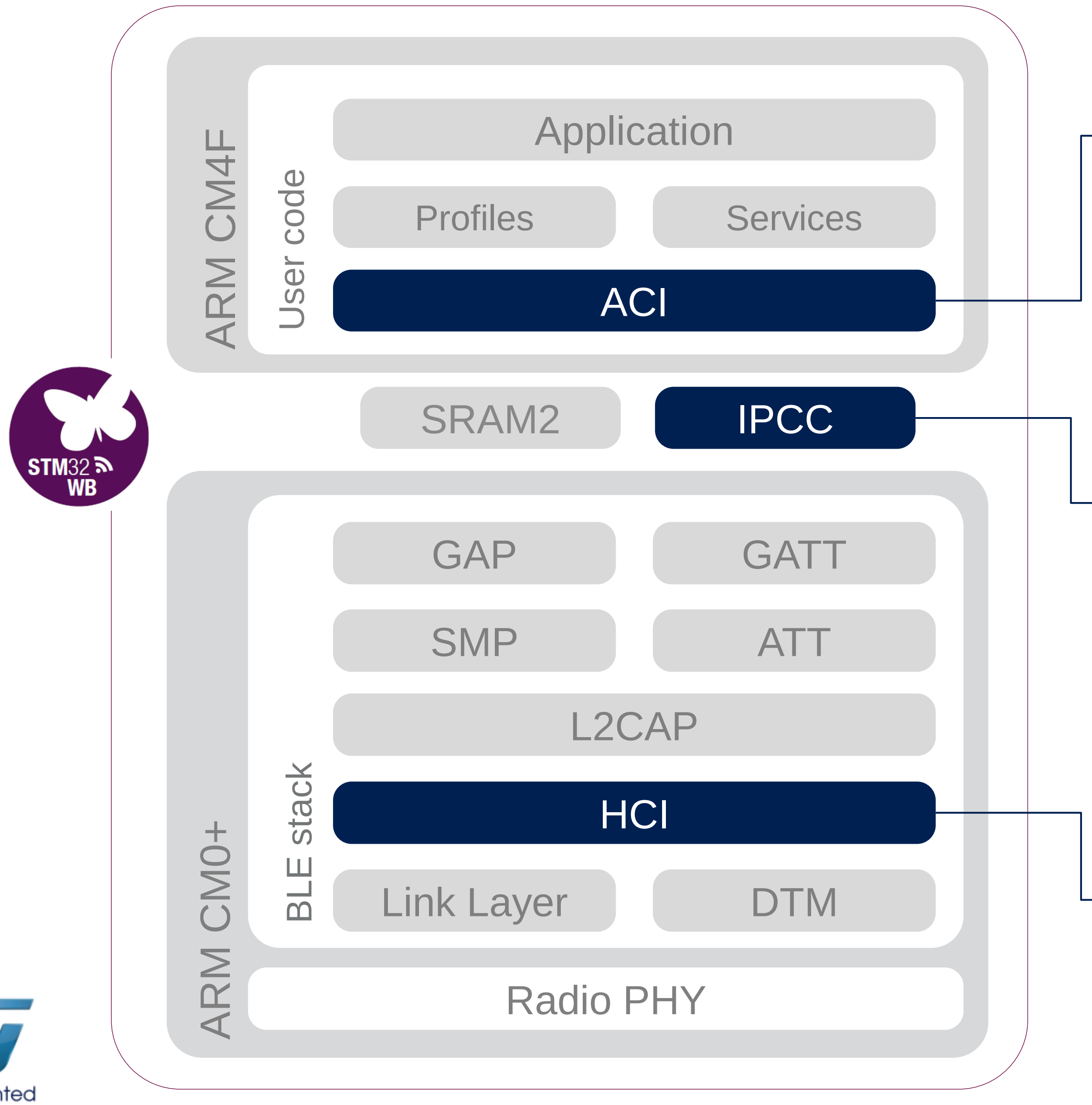
Data fragmentation and reassembly

Routing ATT and SMP

DTM – Direct Test Mode

(more details in **chapter 5**)

Bluetooth LE various stack layers



ACI – Application Command Interface

Set of commands used by the application to control the stack and communicate with it

IPCC – Inter-Processor Communication Controller

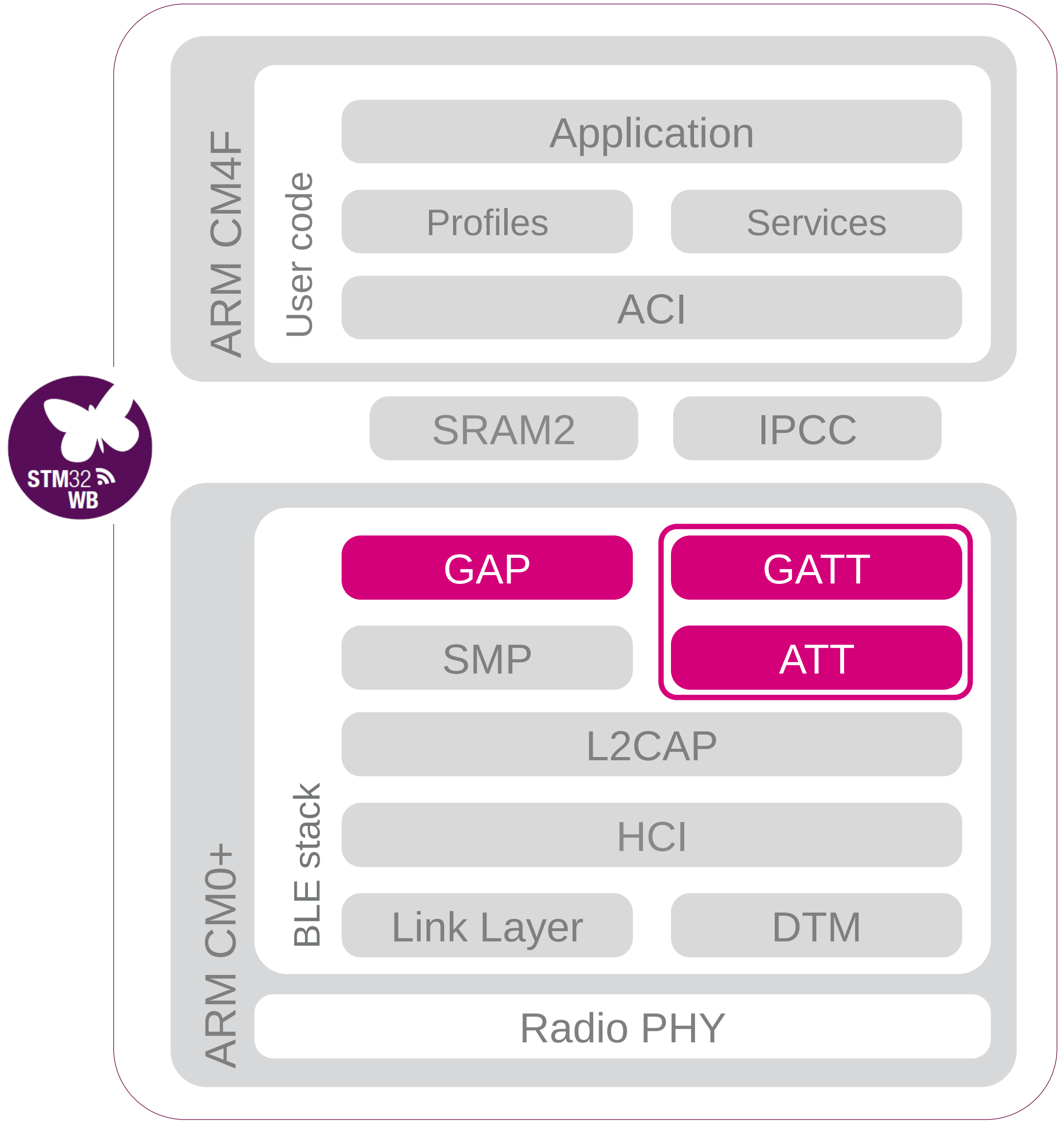
Microcontroller peripheral enabling data exchange between both CPUs

HCI – Host Controller Interface

Interface between the Controller (Link Layer and PHY) and the Host (upper stack layers)

On SoC useful to get the access to lower-layer features

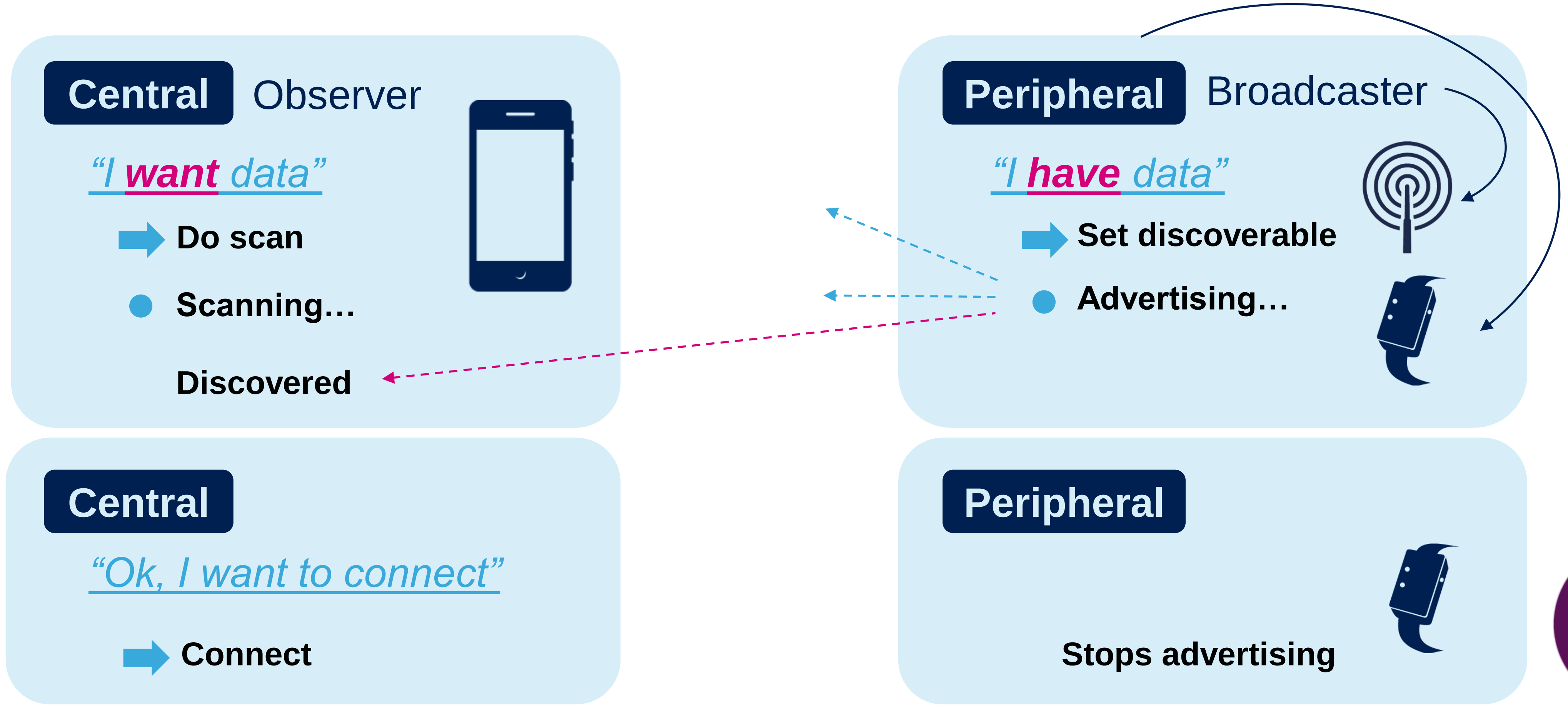
Bluetooth LE various stack layers



Bluetooth LE basic principles

GAP – Generic Access Profile

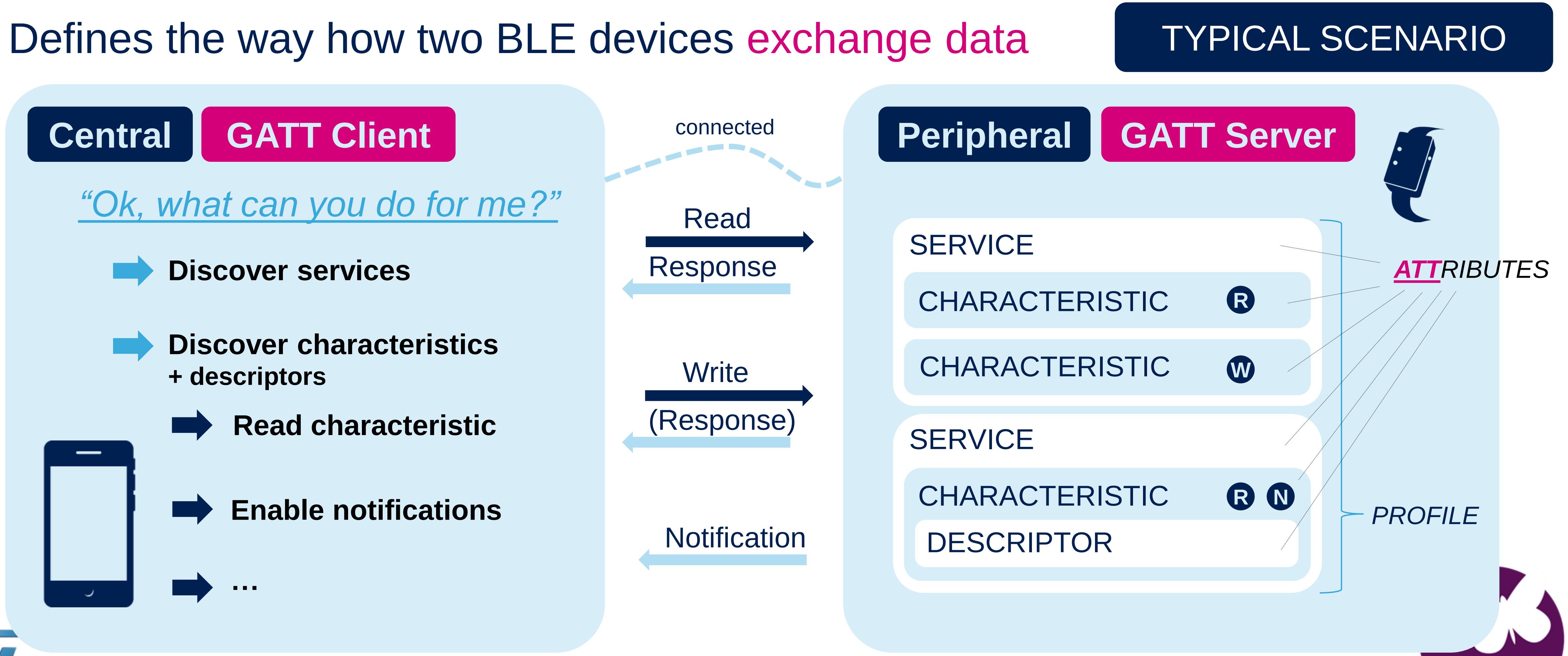
- GAP layer controls advertising and connections (makes a device visible to the outside world)
- Also determines how two devices can interact with each other



Bluetooth LE basic principles

GATT – Generic Attribute Profile

- Defines the way how two BLE devices exchange data



Communication attempt repeated with connection interval defined by central (suggested by peripheral)

Bluetooth LE basic principles

ATT – Attribute Protocol

- Client/server protocol, forms the basis of data exchange in BLE applications
- Server (BLE peripheral) provides data upon request from a client (central device)
- Server data stored in so-called Attribute Table, which contains a series of record (attributes) of various types
- The main types are called Services and Characteristics

ATTRIBUTE =

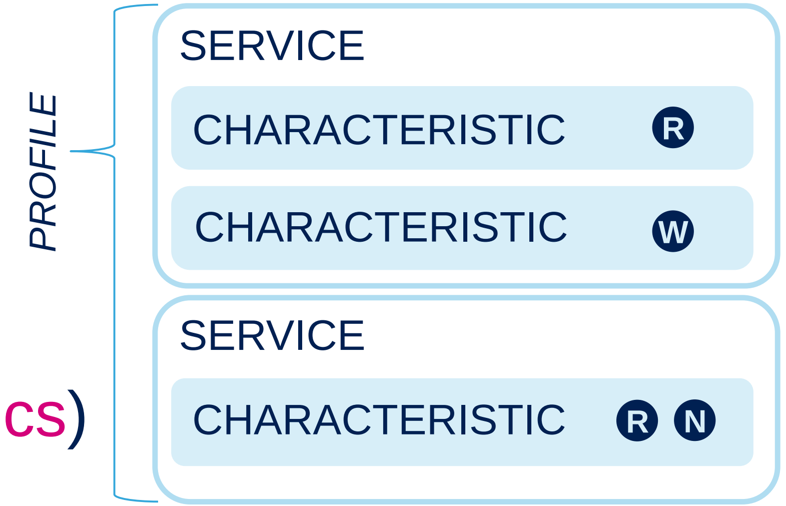
- 16-bit **handle**, an identifier used to access the attribute
- 16-bit or 128-bit **UUID** which defines the attribute **type** and **nature of the data** in the value
- **value** of a certain **length** (bytes)
- permissions (read, write,…)

For ATT just an array of bytes stored in a table, data logic and hierarchy given by GATT and app layer

Bluetooth LE basic principles

GATT – Generic Attribute Profile details

- Comes into play when a connection is established
- Defines data exchange between two BLE devices
- Adds a data model and hierarchy on top of the ATT (by means of concepts called services and characteristics)
- Services organized in GATT profiles
- Each profile can contain multiple services
- Custom profiles vs. adopted profiles by Bluettoth SIG



Bluetooth SIG adopted profiles:
Heart Rate Profile
Fitness Machine Profile
Location and Navigation Profile
...

Bluetooth LE basic principles

GATT services and characteristics

- A service is a container for logically related data items
  (e.g. *Device Information Service* – various information about the device)

- Characteristics are logically related data items within one service
  (e.g. *Serial Number String* and *Manufacturer Name String* from the *Device Information Service*)

- A characteristic consists of a type, a value, some properties, permissions and optionally descriptors

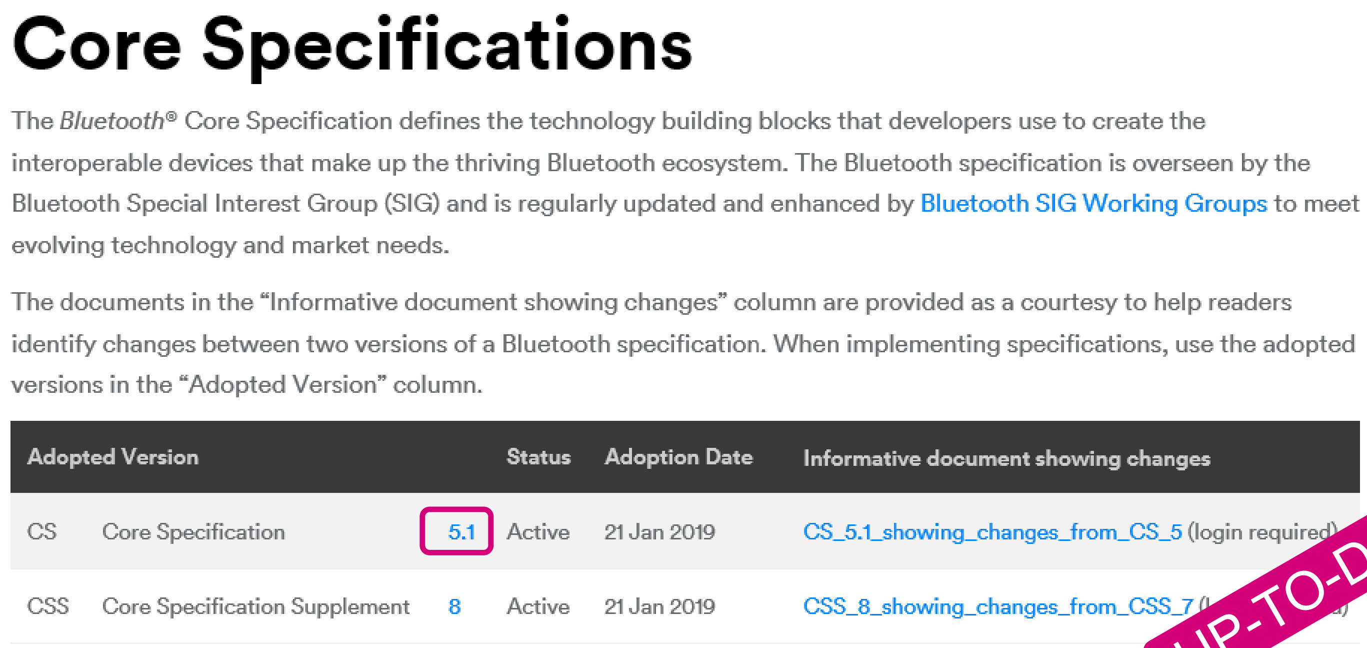
- Descriptors either provides additional details or allows configuration of behavior related to the characteristic (e.g. turn on notifications)

Where to go next?

1

Bluetooth® https://www.bluetooth.com/

https://www.bluetooth.com/specifications/bluetooth-core-specification



Core Specifications

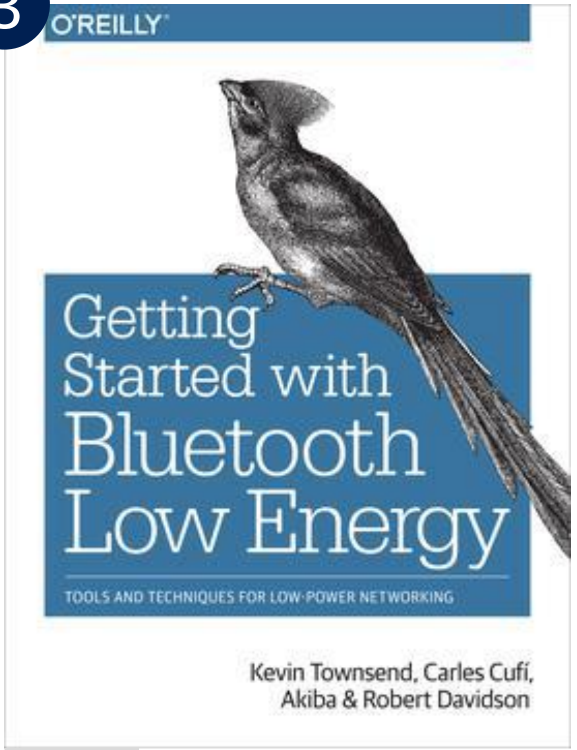
The *Bluetooth*® Core Specification defines the technology building blocks that developers use to create the interoperable devices that make up the thriving Bluetooth ecosystem. The Bluetooth specification is overseen by the Bluetooth Special Interest Group (SIG) and is regularly updated and enhanced by Bluetooth SIG Working Groups to meet evolving technology and market needs.

The documents in the "Informative document showing changes" column are provided as a courtesy to help readers identify changes between two versions of a Bluetooth specification. When implementing specifications, use the adopted versions in the "Adopted Version" column.

Adopted Version			Status	Adoption Date	Informative document showing changes
CS	Core Specification	5.1	Active	21 Jan 2019	CS_5.1_showing_changes_from_CS_5 (login required)
CSS	Core Specification Supplement	8	Active	21 Jan 2019	CSS_8_showing_changes_from_CSS_7 (l[illegible]d)

UP-TO-DATE !

3



O'REILLY

Getting Started with Bluetooth Low Energy

TOOLS AND TECHNIQUES FOR LOW-POWER NETWORKING

Kevin Townsend, Carles Cufí, Akiba & Robert Davidson

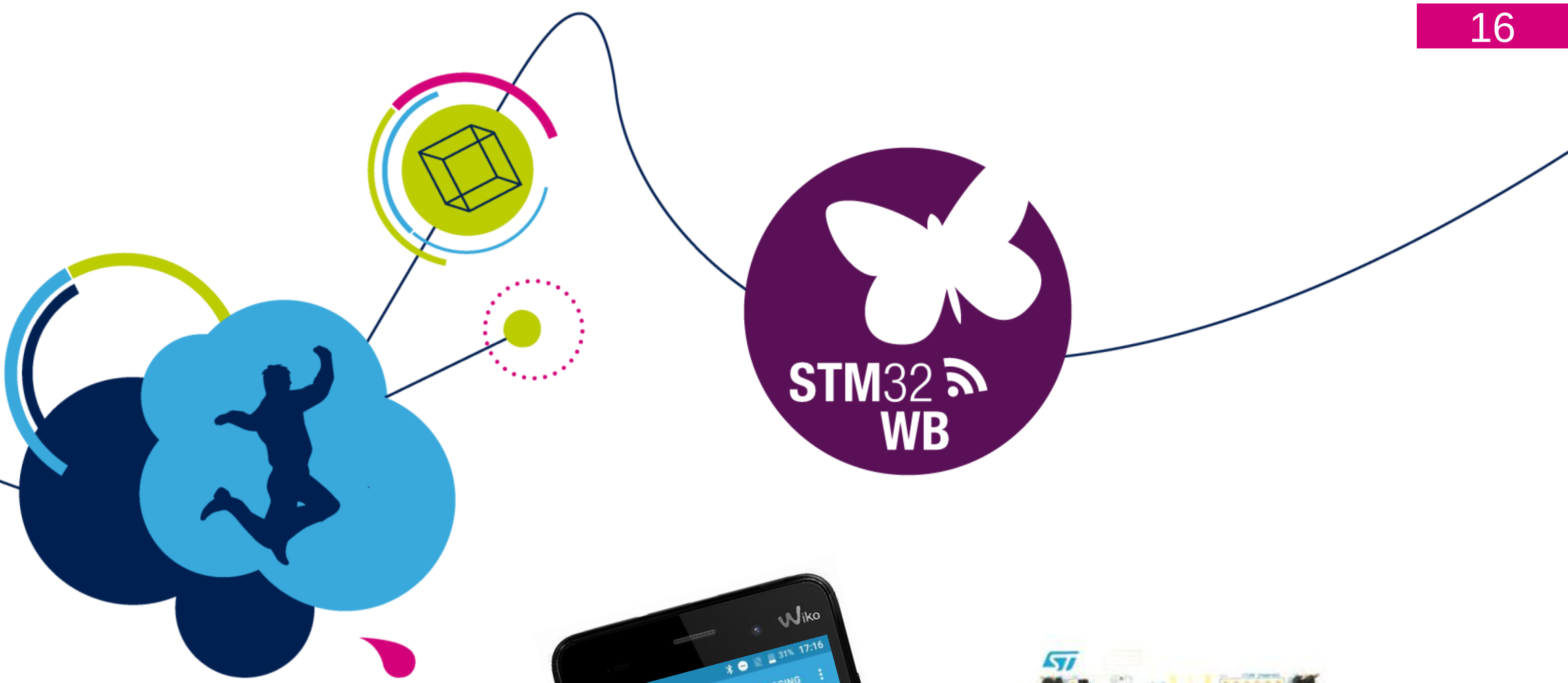
2014

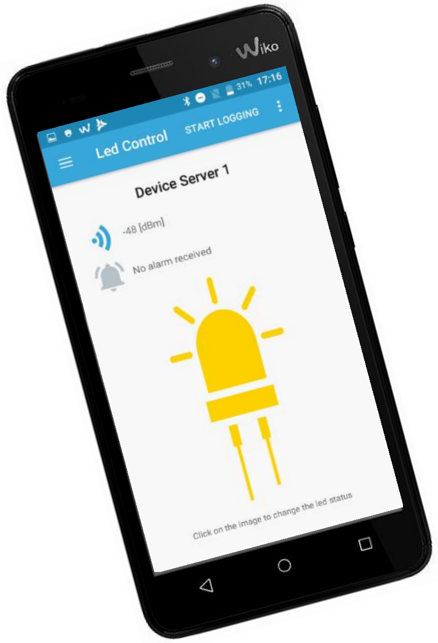
2

Many community websites, e.g.: [Introduction to Bluetooth Low Energy (@adafruit)](#)

life.augmented

STM32 WB

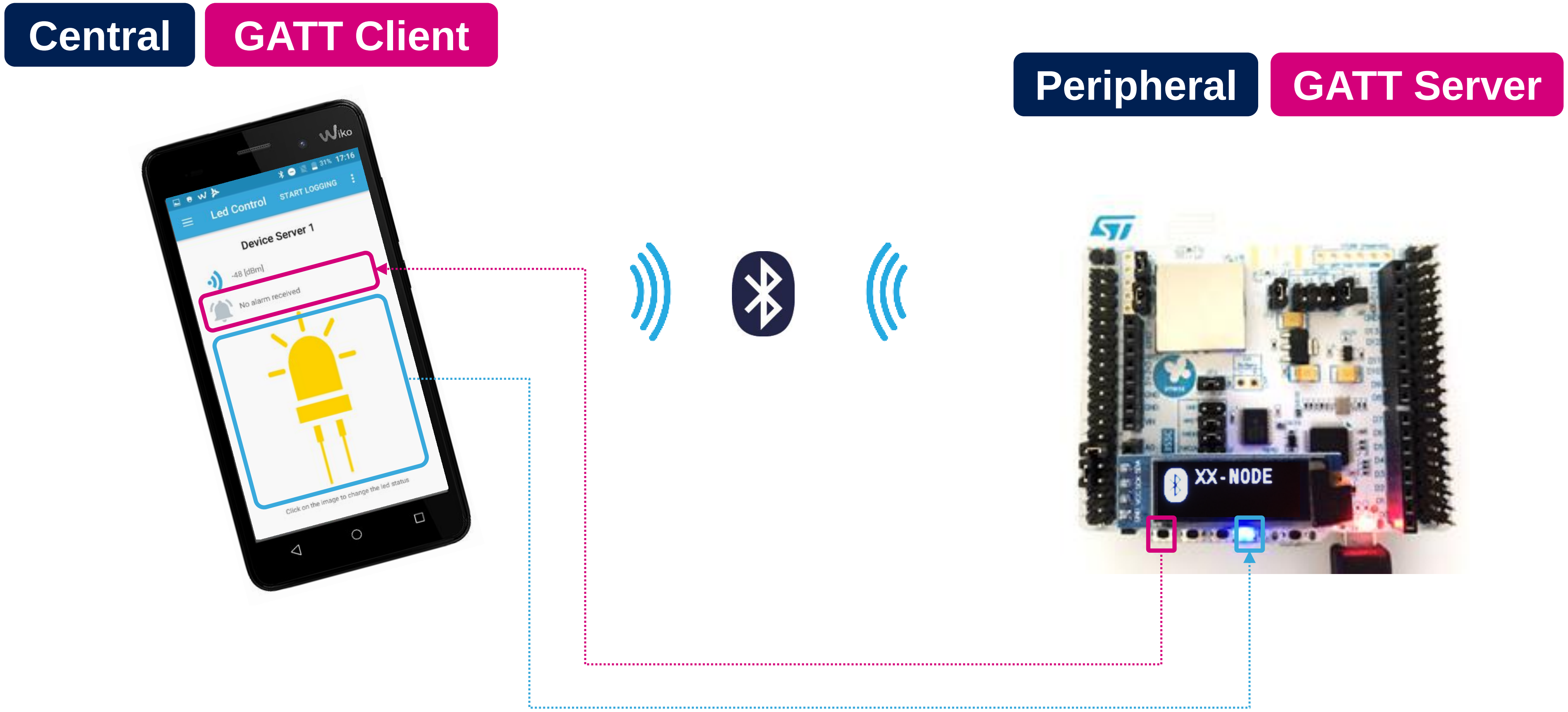




Hands-On #2

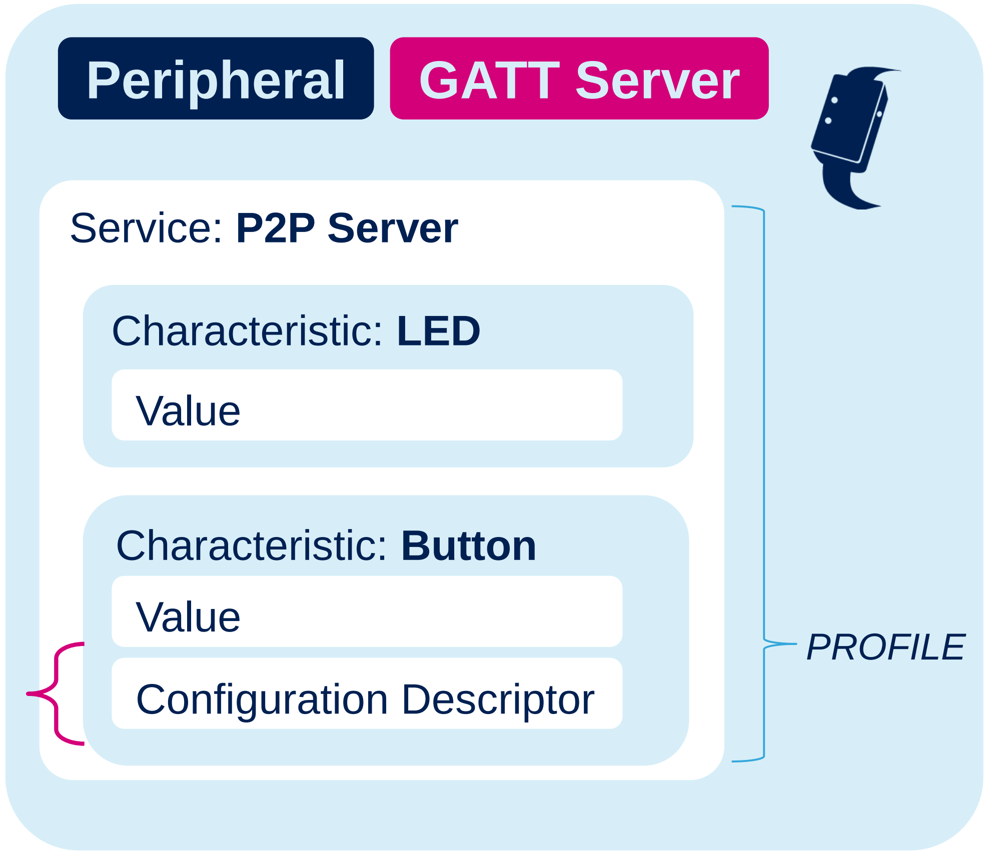
life.augmented

Target


Central
GATT Client
Peripheral
GATT Server
Led Control
START LOGGING
Device Server 1
-48 [dBm]
No alarm received
Click on the image to change the led status
XX-NODE

Bluetooth LE basic principles

GATT – Generic Attribute Profile

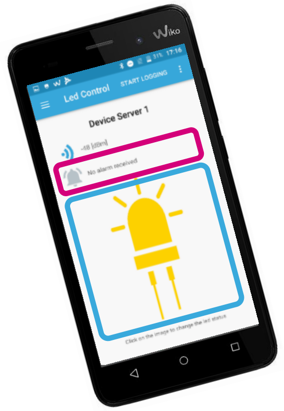


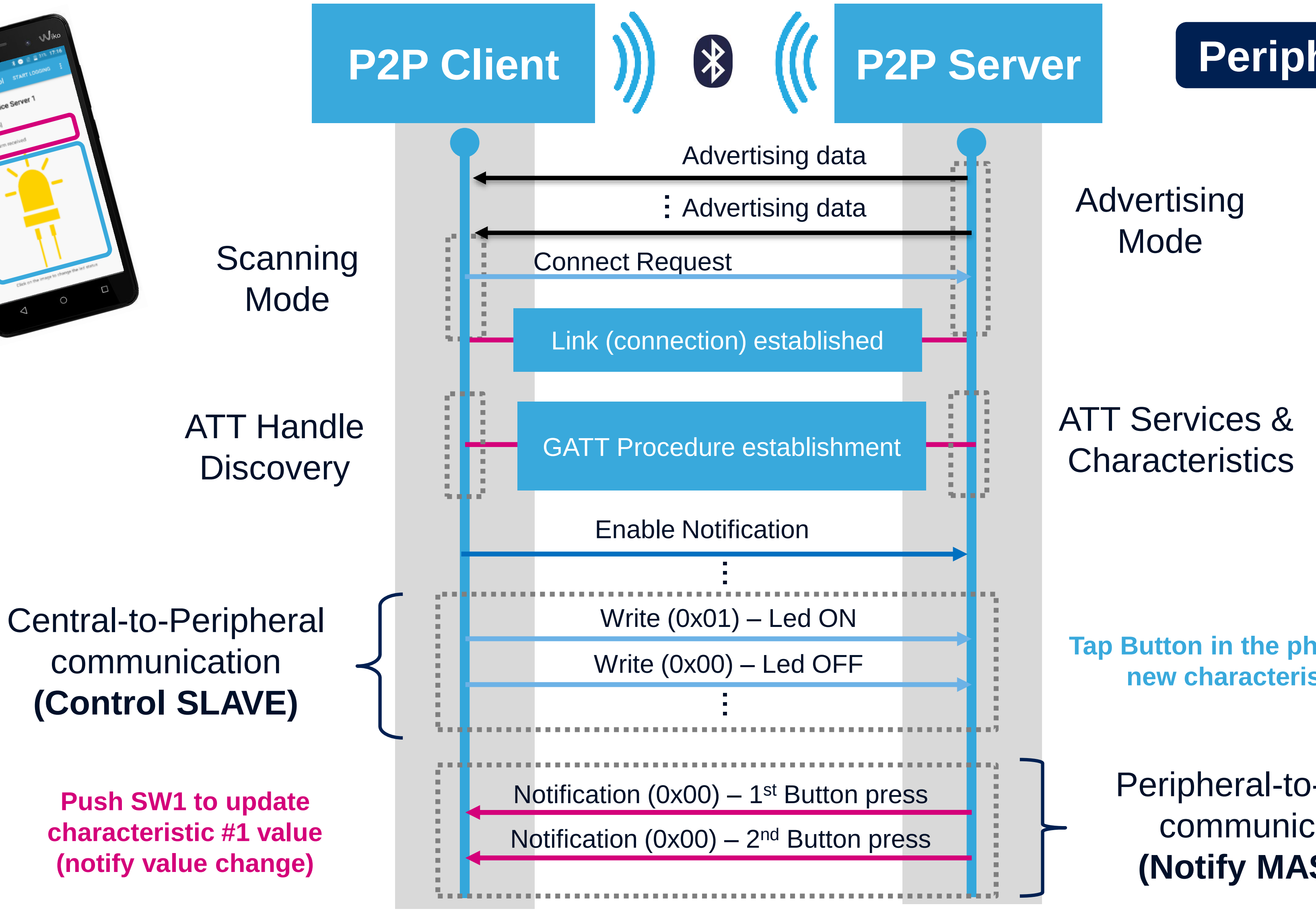
Used in this case by client to enable notification on Button characteristic value change

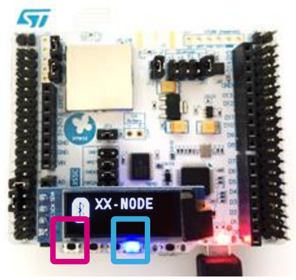
How should it work?

Central **GATT Client**

Peripheral **GATT Server**




P2P Client
P2P Server
Advertising data
Advertising data
Connect Request
Link (connection) established
GATT Procedure establishment
Enable Notification
Write (0x01) – Led ON
Write (0x00) – Led OFF
Notification (0x00) – 1st Button press
Notification (0x00) – 2nd Button press
Scanning Mode
Advertising Mode
ATT Handle Discovery
ATT Services & Characteristics
Central-to-Peripheral communication
(Control SLAVE)
Peripheral-to-Central communication
(Notify MASTER)



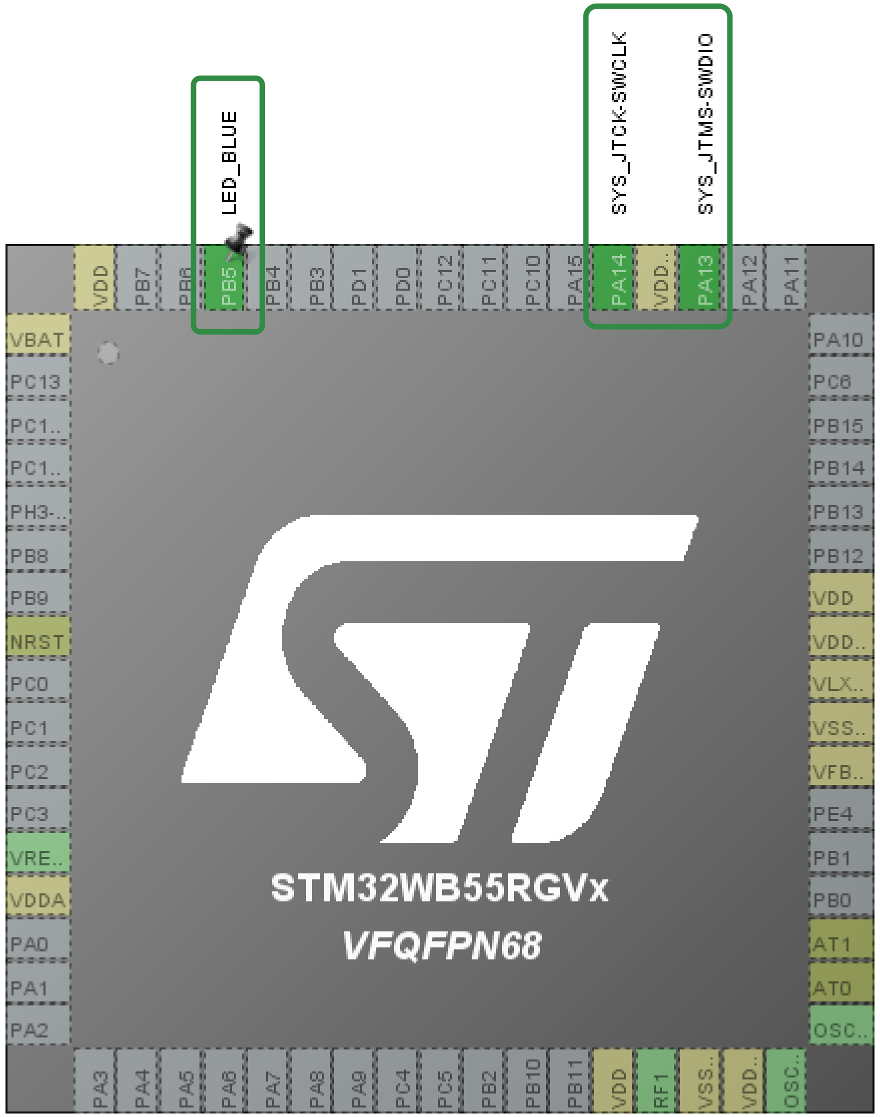
Let’s start with STM32CubeMX

Tap Button in the phone app to write new characteristic #2 value

Push SW1 to update characteristic #1 value (notify value change)

Starting point

SWCLK @PA14
SWDIO @PA13
LED_BLUE @PB5



HandsOn_1

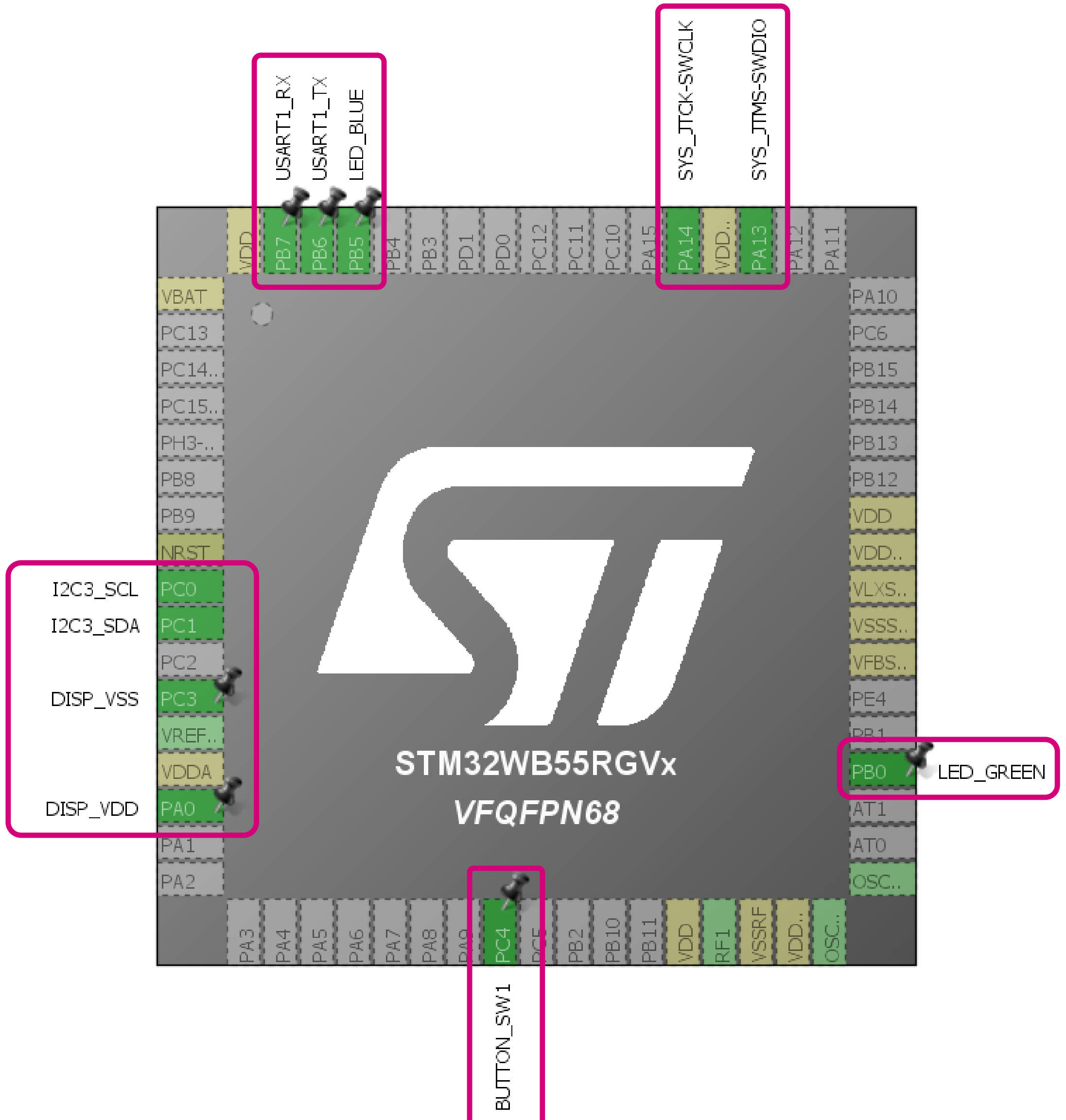
Let’s do a small time shift!

Move to slide **30**!!!

Let’s continue from this state

Open C:\STM32WB_workshop\HandsOns\HandsOn_2\HandsOn_2.ioc

Move to this slide (Slide **30**)

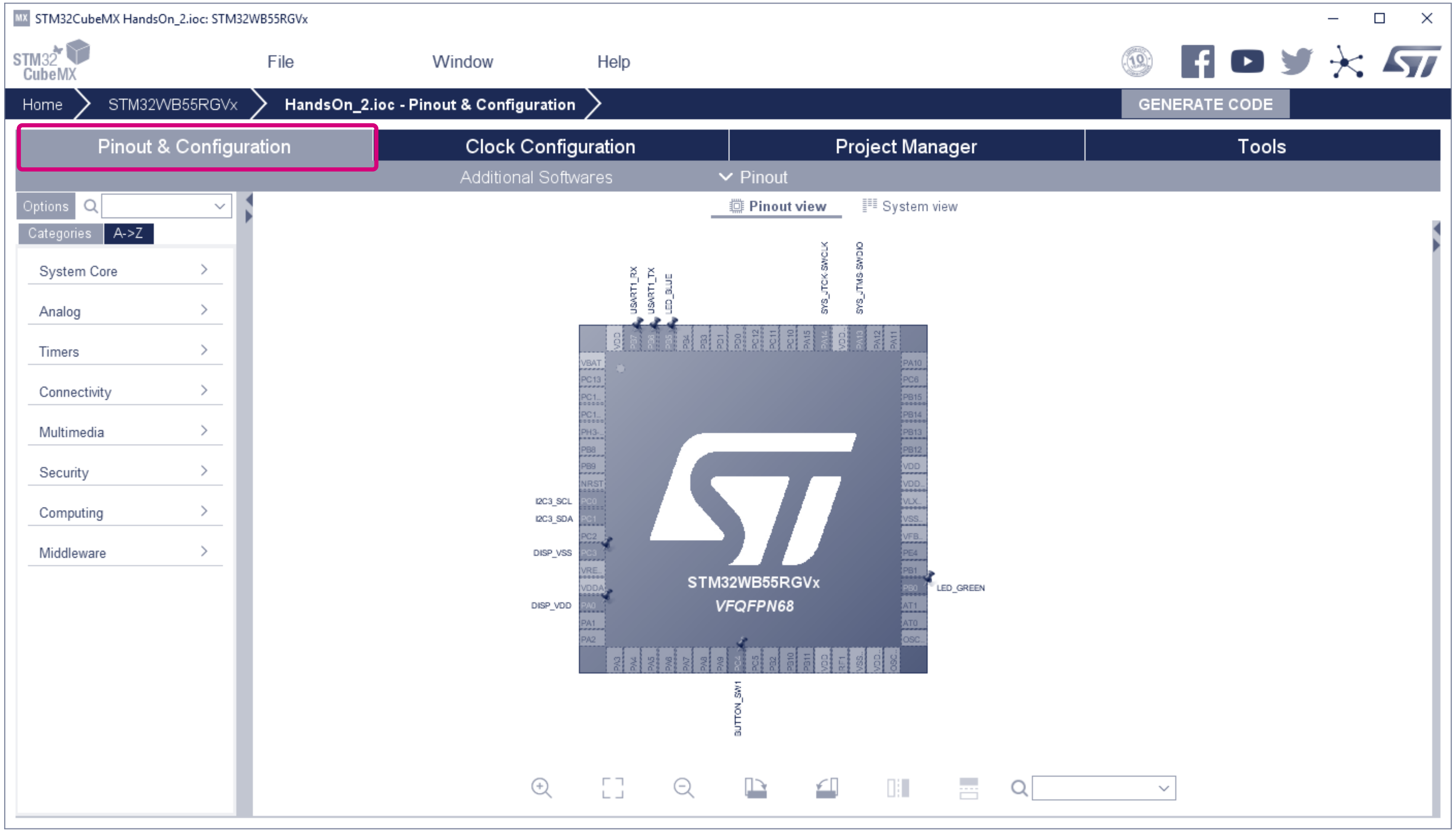


HandsOn_2

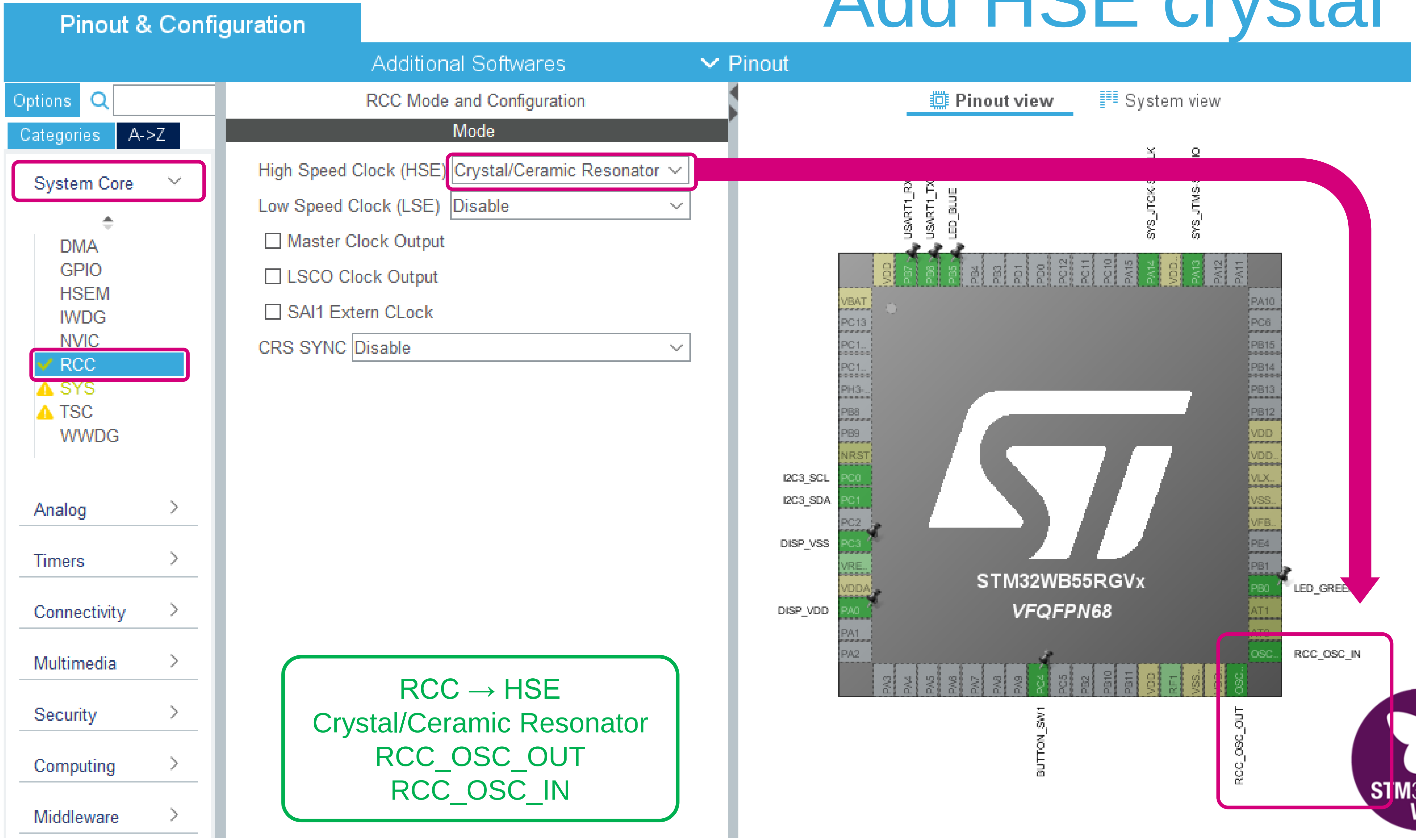
HandsOn_2.ioc

I2C3_SCL @PC0
I2C3_SDA @PC1
DISP_VSS @PC3
DISP_VDD @PA0
LED_GREEN @PB0
LED_BLUE @PB5
SWD interface

Continue with Pinout Configuration



Add HSE crystal



Pinout & Configuration

Additional Softwares ⌄ Pinout

Options | Categories | A->Z

System Core: DMA, GPIO, HSEM, IWDG, NVIC, ✓ RCC, ⚠ SYS, ⚠ TSC, WWDG

Analog, Timers, Connectivity, Multimedia, Security, Computing, Middleware

RCC Mode and Configuration

Mode

High Speed Clock (HSE): Crystal/Ceramic Resonator

Low Speed Clock (LSE): Disable

☐ Master Clock Output

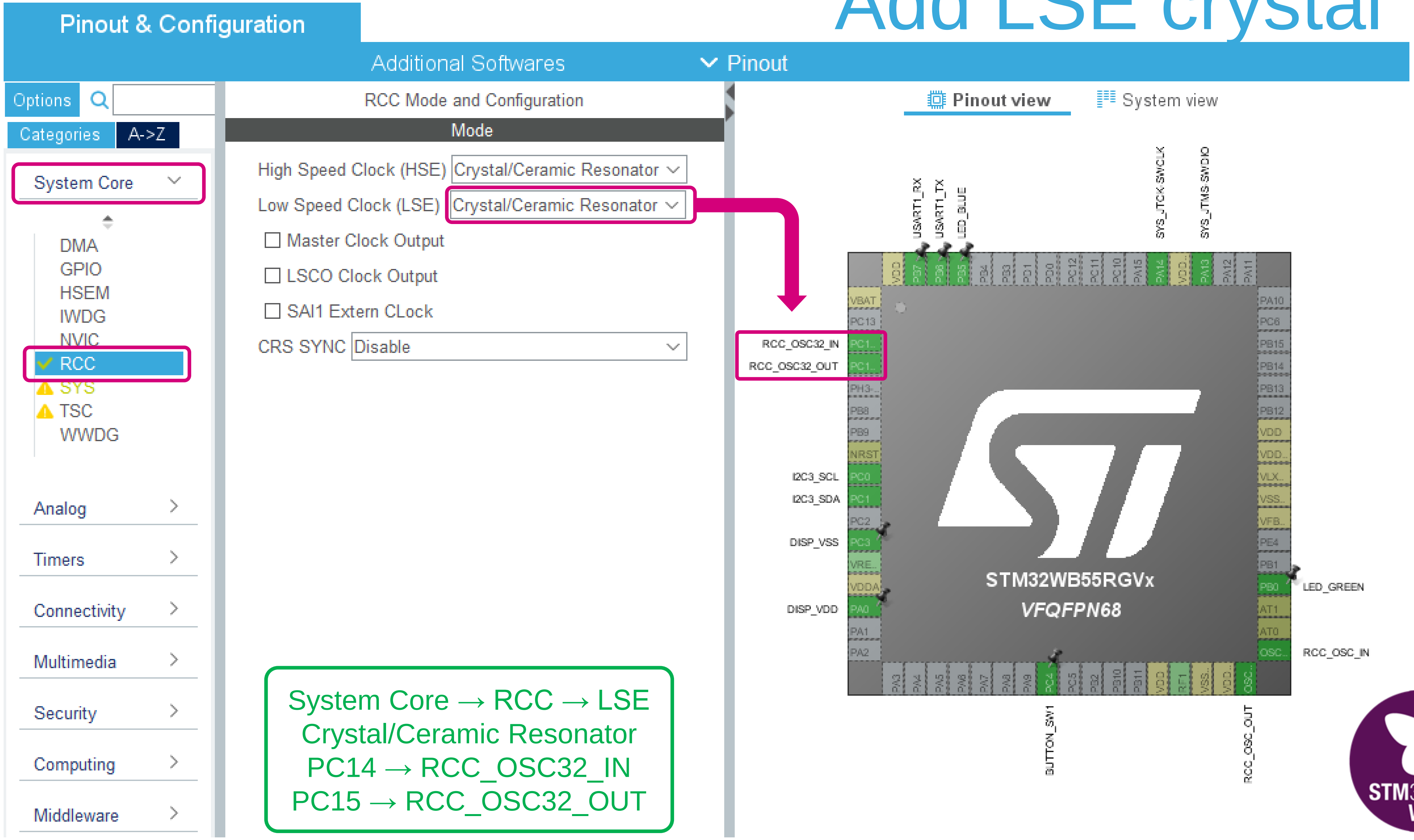
☐ LSCO Clock Output

☐ SAI1 Extern CLock

CRS SYNC: Disable

RCC → HSE
Crystal/Ceramic Resonator
RCC_OSC_OUT
RCC_OSC_IN

Add LSE crystal



High Speed Clock (HSE) Crystal/Ceramic Resonator

Low Speed Clock (LSE) Crystal/Ceramic Resonator

☐ Master Clock Output

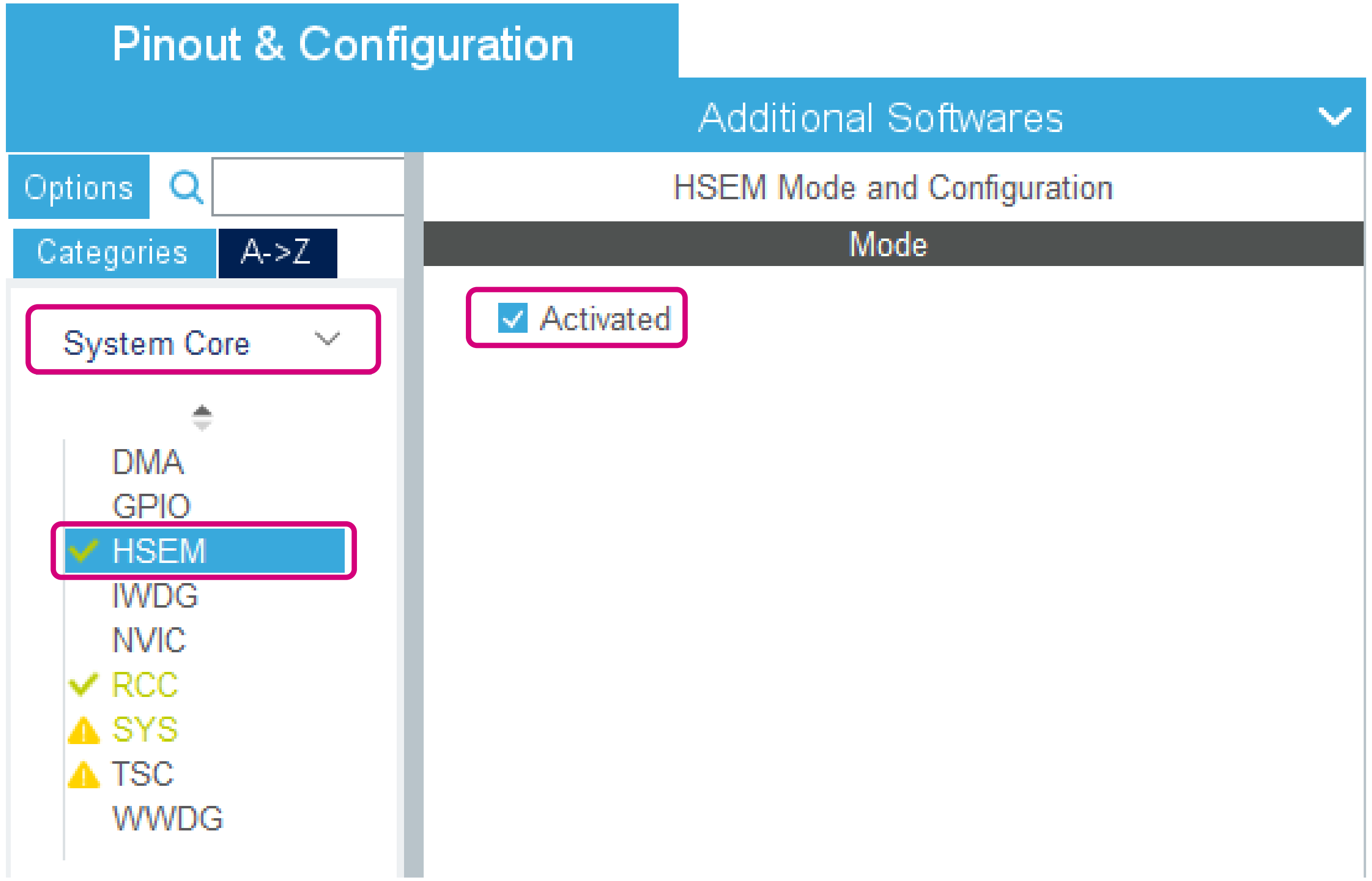
☐ LSCO Clock Output

☐ SAI1 Extern CLock

CRS SYNC Disable

System Core → RCC → LSE
Crystal/Ceramic Resonator
PC14 → RCC_OSC32_IN
PC15 → RCC_OSC32_OUT

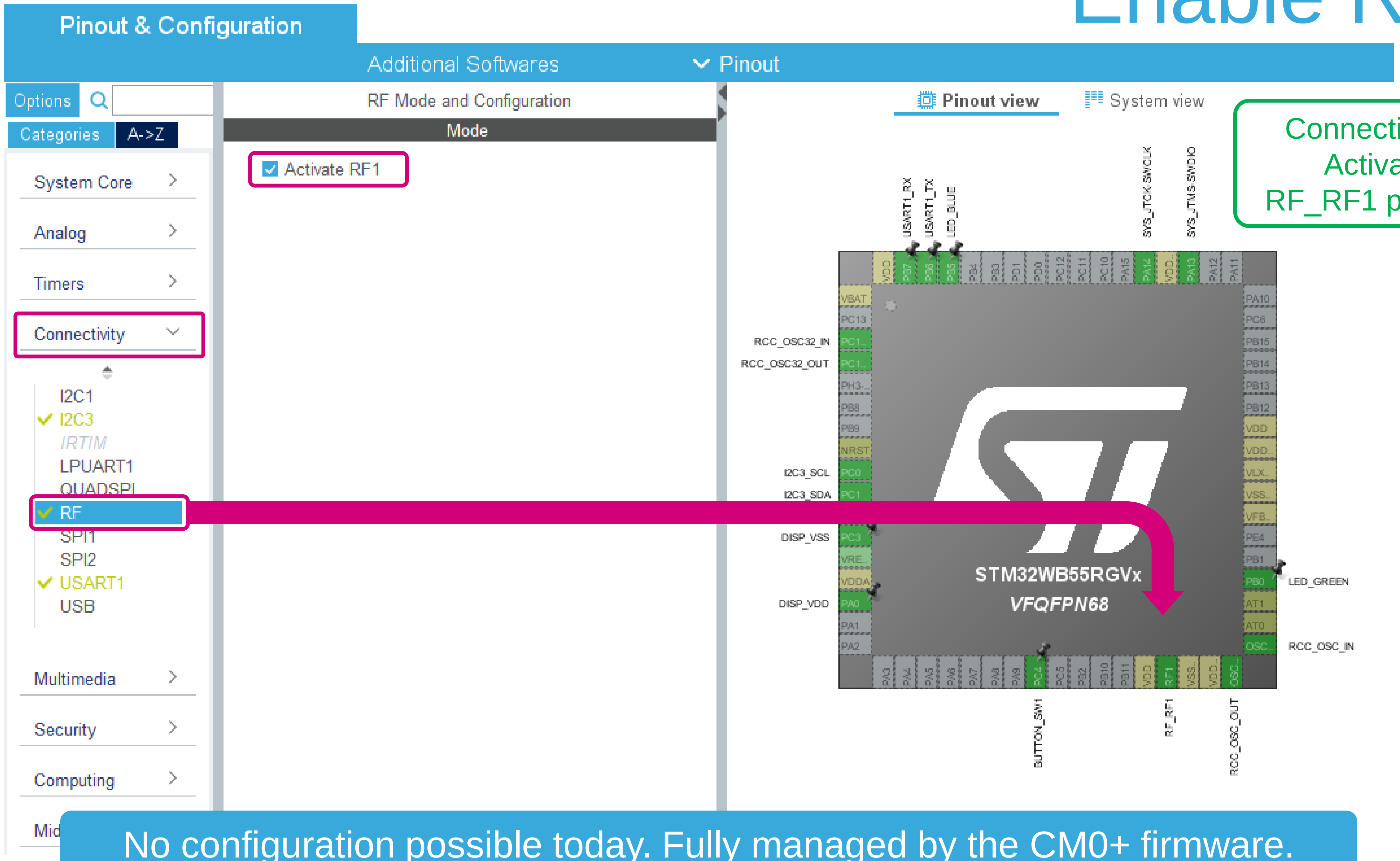
Enable Hardware Semaphores (HSEM)



System Core → HSEM Activated

No configuration possible today. Fully managed by the middleware. Just to keep in mind that it exists and is in use to manage access to resources shared by both CM0+ and CM4F.

Enable RF



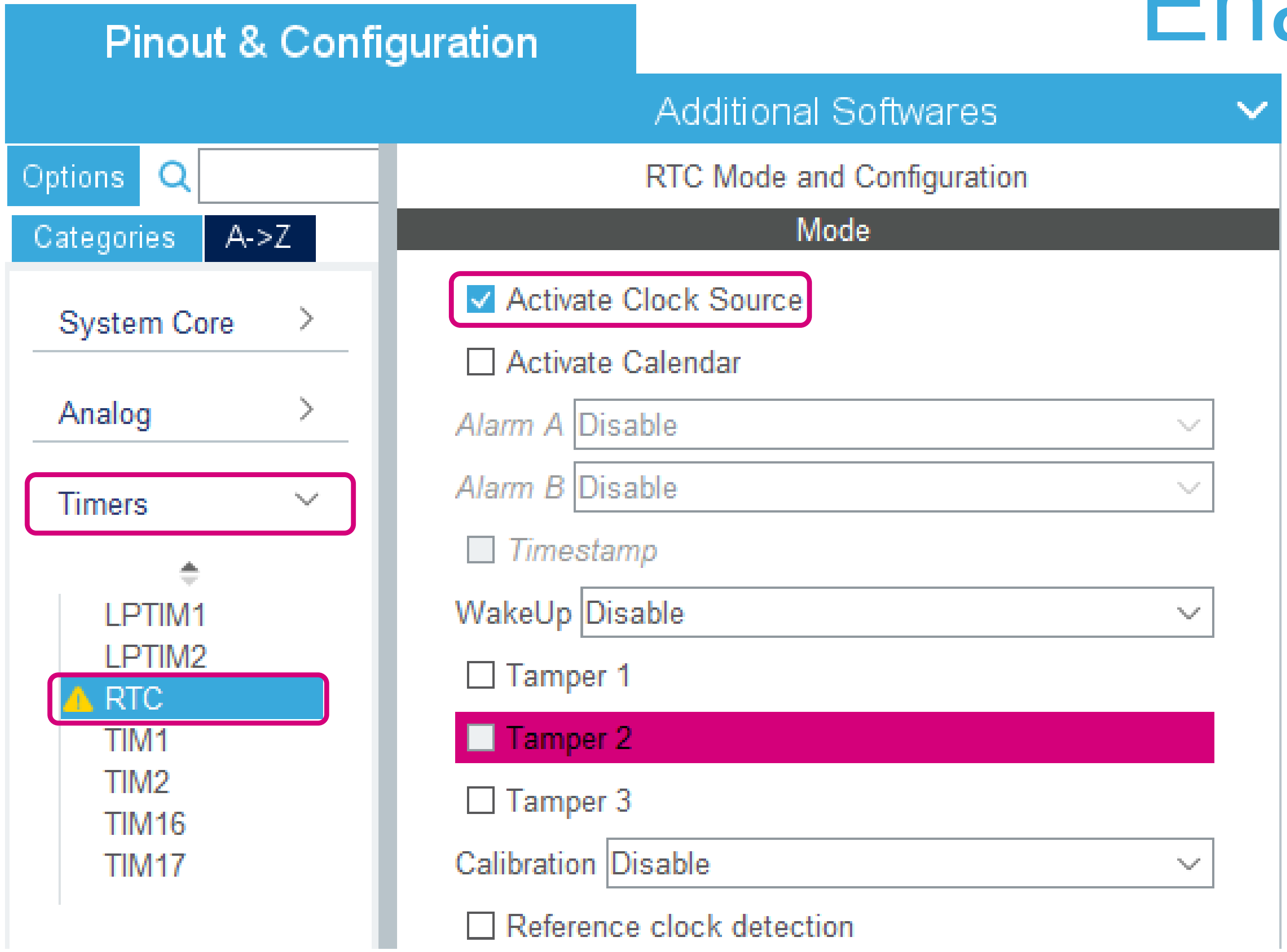
☑ Activate RF1

Connectivity → RF
Activate RF1
RF_RF1 pin assigned

No configuration possible today. Fully managed by the CM0+ firmware.

Enable RTC

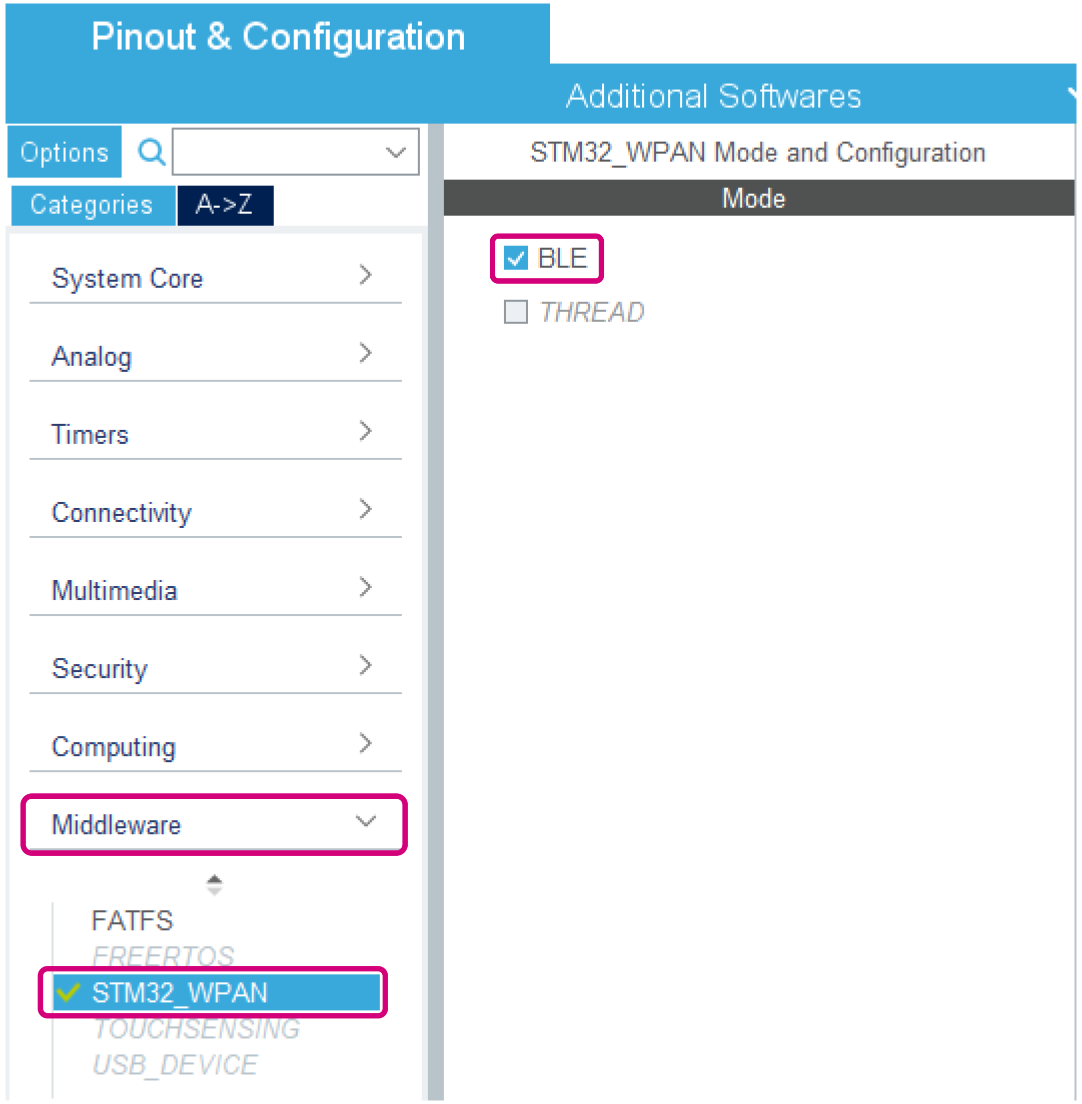
Timers → RTC
Activate Clock Source



- [✓] Activate Clock Source
- [ ] Activate Calendar
- Alarm A: Disable
- Alarm B: Disable
- [ ] Timestamp
- WakeUp: Disable
- [ ] Tamper 1
- [ ] Tamper 2
- [ ] Tamper 3
- Calibration: Disable
- [ ] Reference clock detection

RTC used by STM32_WPAN middleware just to provide some timebase for SW timers and Low-Power modes support. Fully modifiable according to user application needs.

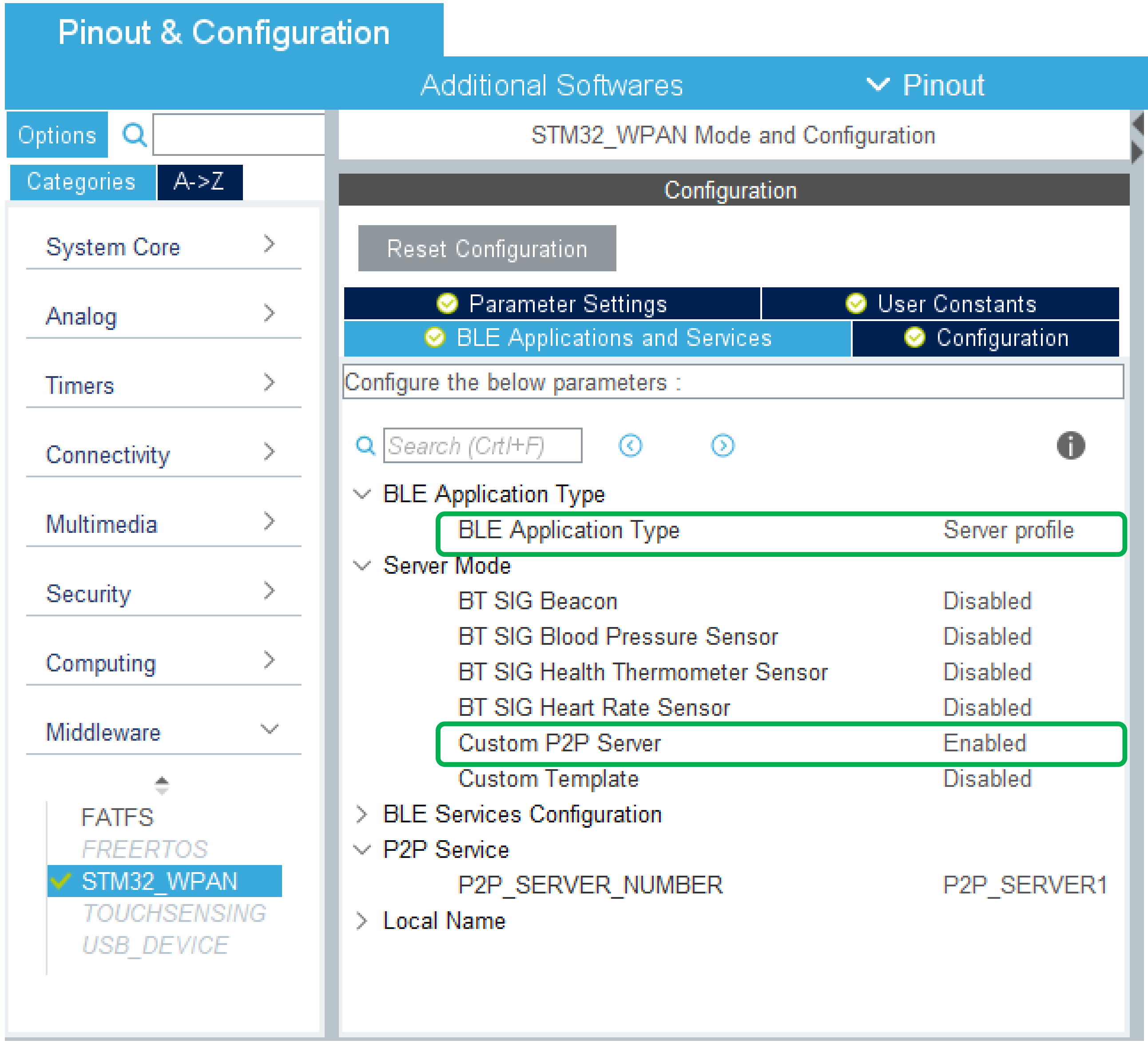
Enable STM32_WPAN BLE Middleware



[✓] BLE
[ ] THREAD

Middleware → STM32_WPAN BLE

Configure STM32_WPAN BLE Middleware



Configure the below parameters :

```
BLE Application Type
    BLE Application Type                 Server profile
Server Mode
    BT SIG Beacon                        Disabled
    BT SIG Blood Pressure Sensor         Disabled
    BT SIG Health Thermometer Sensor     Disabled
    BT SIG Heart Rate Sensor             Disabled
    Custom P2P Server                    Enabled
    Custom Template                      Disabled
BLE Services Configuration
P2P Service
    P2P_SERVER_NUMBER                    P2P_SERVER1
Local Name
```

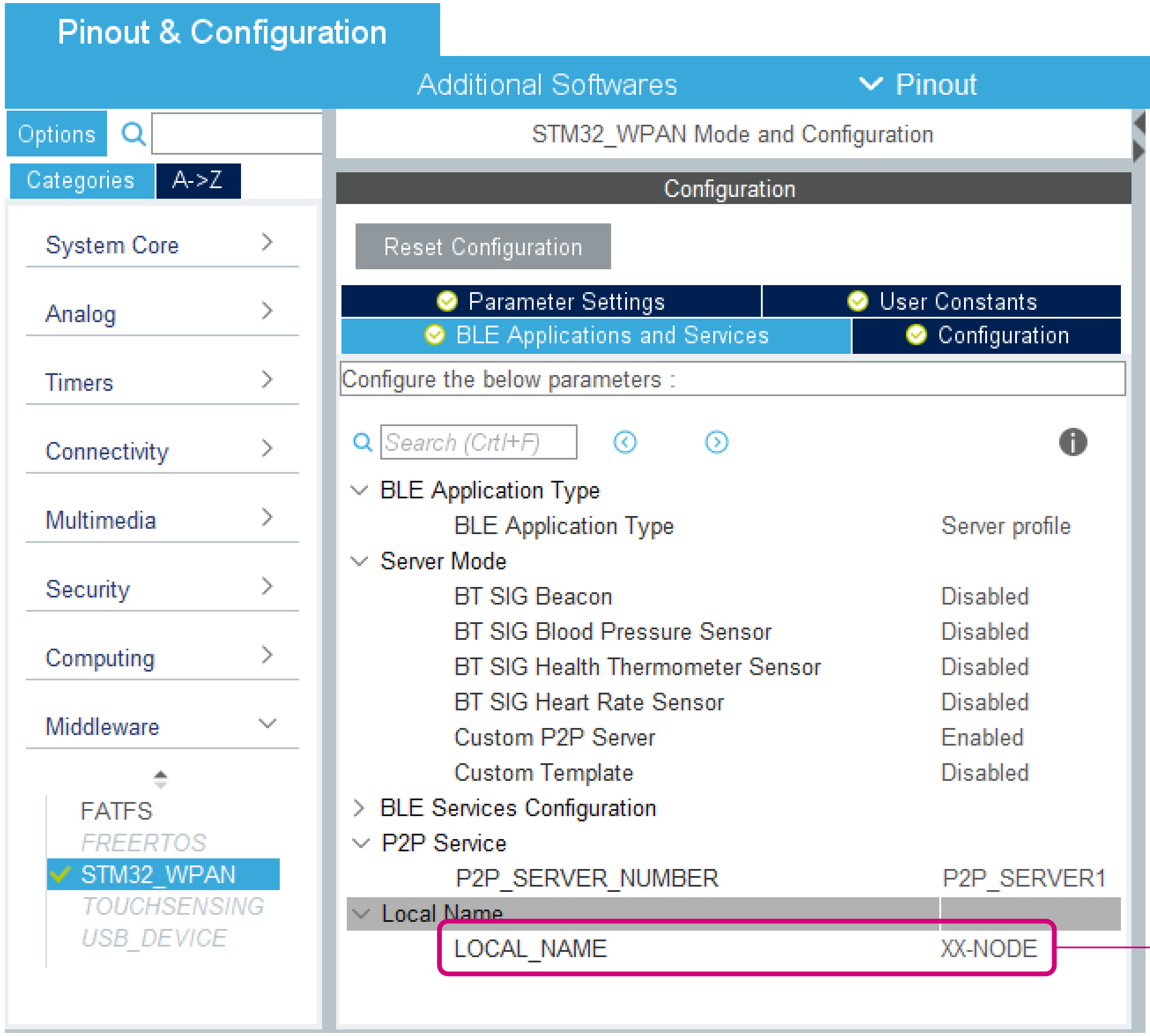
BLE Application Type → Server profile
Server Mode → Custom P2P Server Enabled

No need to change, already pre-configured.

Your very own happy number for today

01, 02,…. XX
To be used in your advertised complete local name.

Configure STM32_WPAN BLE Middleware



Configure the below parameters :

```
BLE Application Type
    BLE Application Type                 Server profile
Server Mode
    BT SIG Beacon                        Disabled
    BT SIG Blood Pressure Sensor         Disabled
    BT SIG Health Thermometer Sensor     Disabled
    BT SIG Heart Rate Sensor             Disabled
    Custom P2P Server                    Enabled
    Custom Template                      Disabled
BLE Services Configuration
P2P Service
    P2P_SERVER_NUMBER                    P2P_SERVER1
Local Name
    LOCAL_NAME                           XX-NODE
```

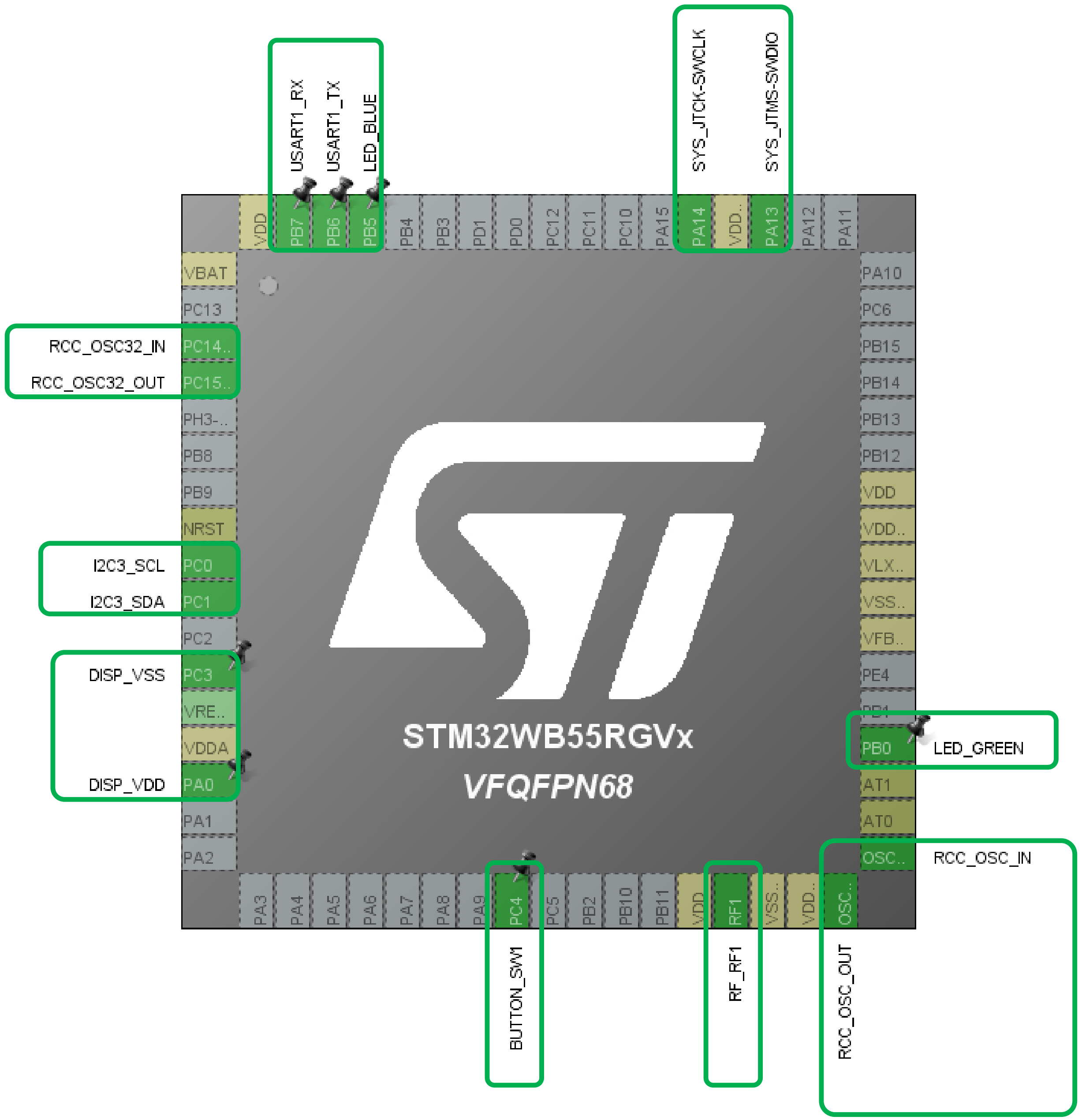
LOCAL_NAME changed according to your happy number

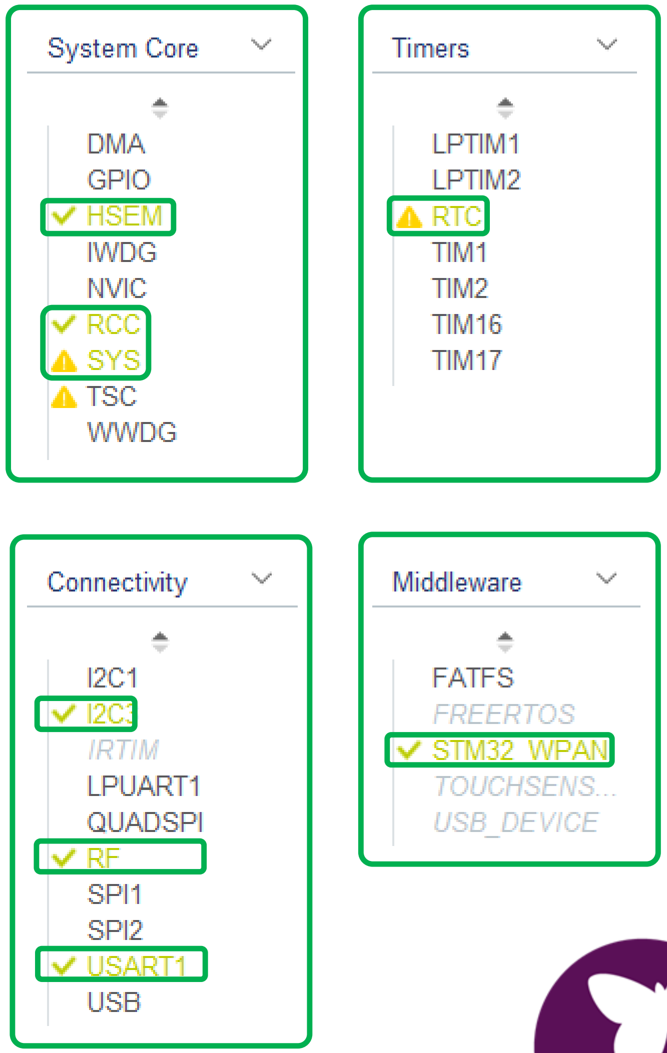
Max number of characters is set to 7 to be aligned with overall length of advertising data used by the generated code.

XX-NODE where **XX** is your happy number

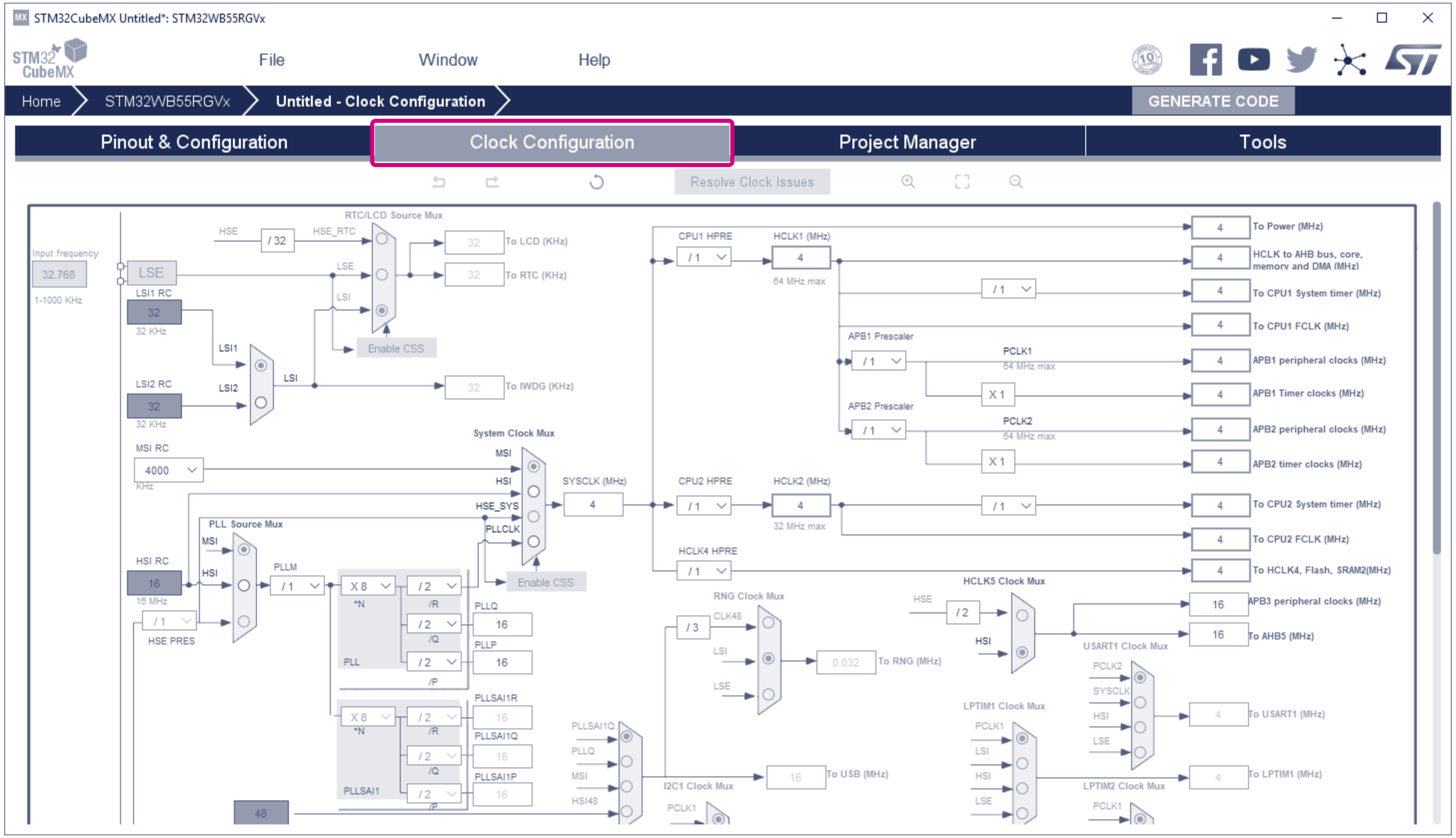
Checkpoint

SWD @PA13/PA14
I2C3_SCL @PC0
I2C3_SDA @PC1
DISP_VSS @PC3
DISP_VDD @PA0
LED_GREEN @PB0
LED_BLUE @PB5
USART1_RX @PB7
USART1_TX @PB6
HSE
LSE
RTC Activated
HSEM Activated
RF Activated
STM32_WPAN BLE



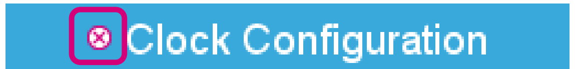


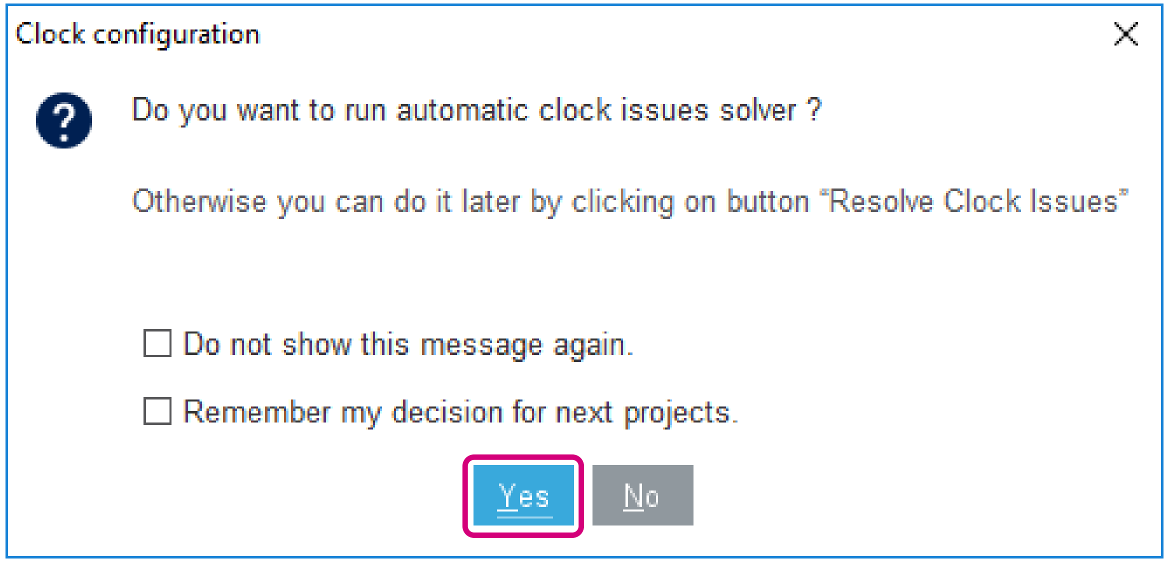
Switch to clock configuration



STM32 WB

Clock issue solver




Clock configuration

Do you want to run automatic clock issues solver ?

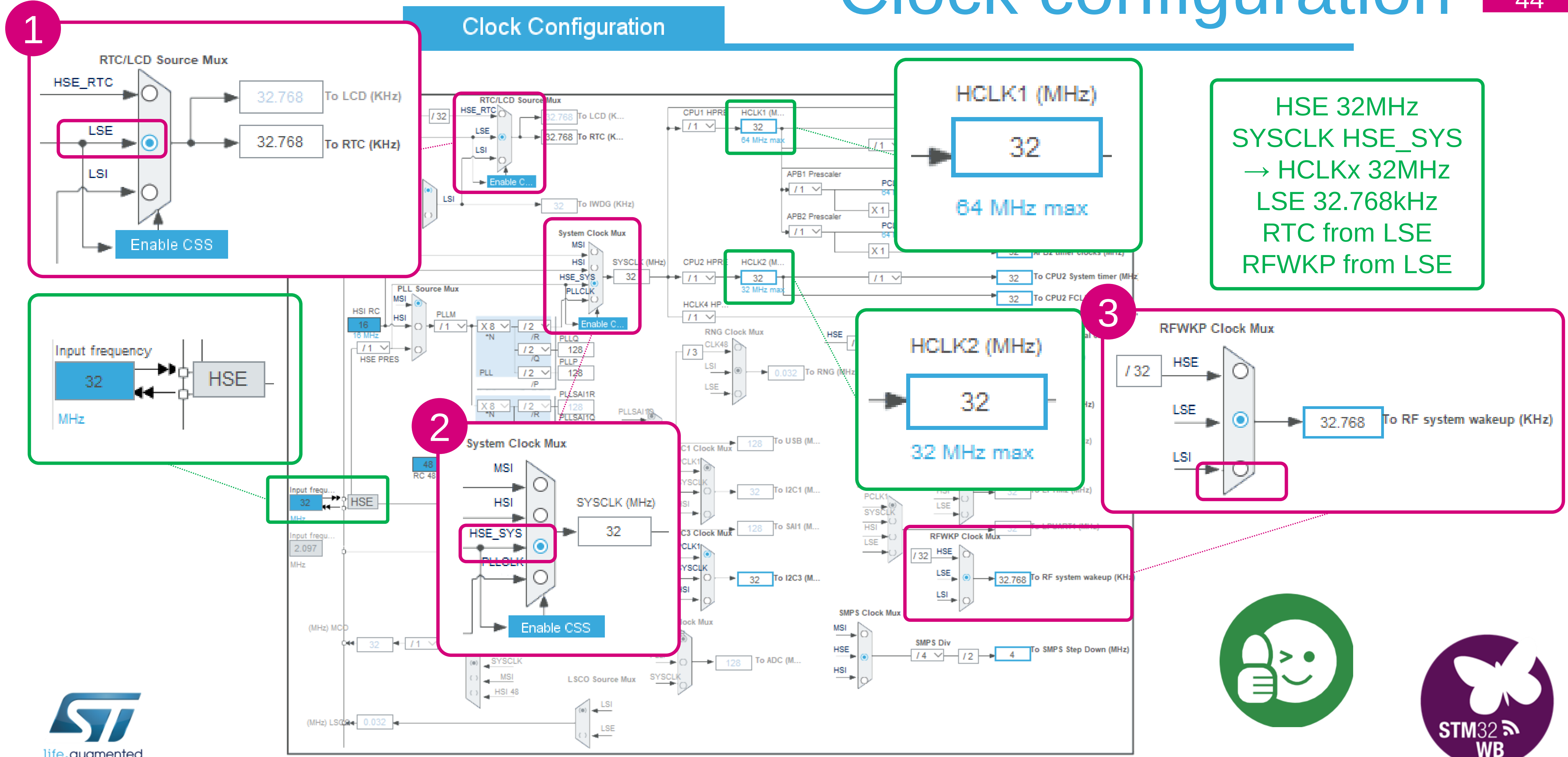
Otherwise you can do it later by clicking on button "Resolve Clock Issues"

☐ Do not show this message again.

☐ Remember my decision for next projects.

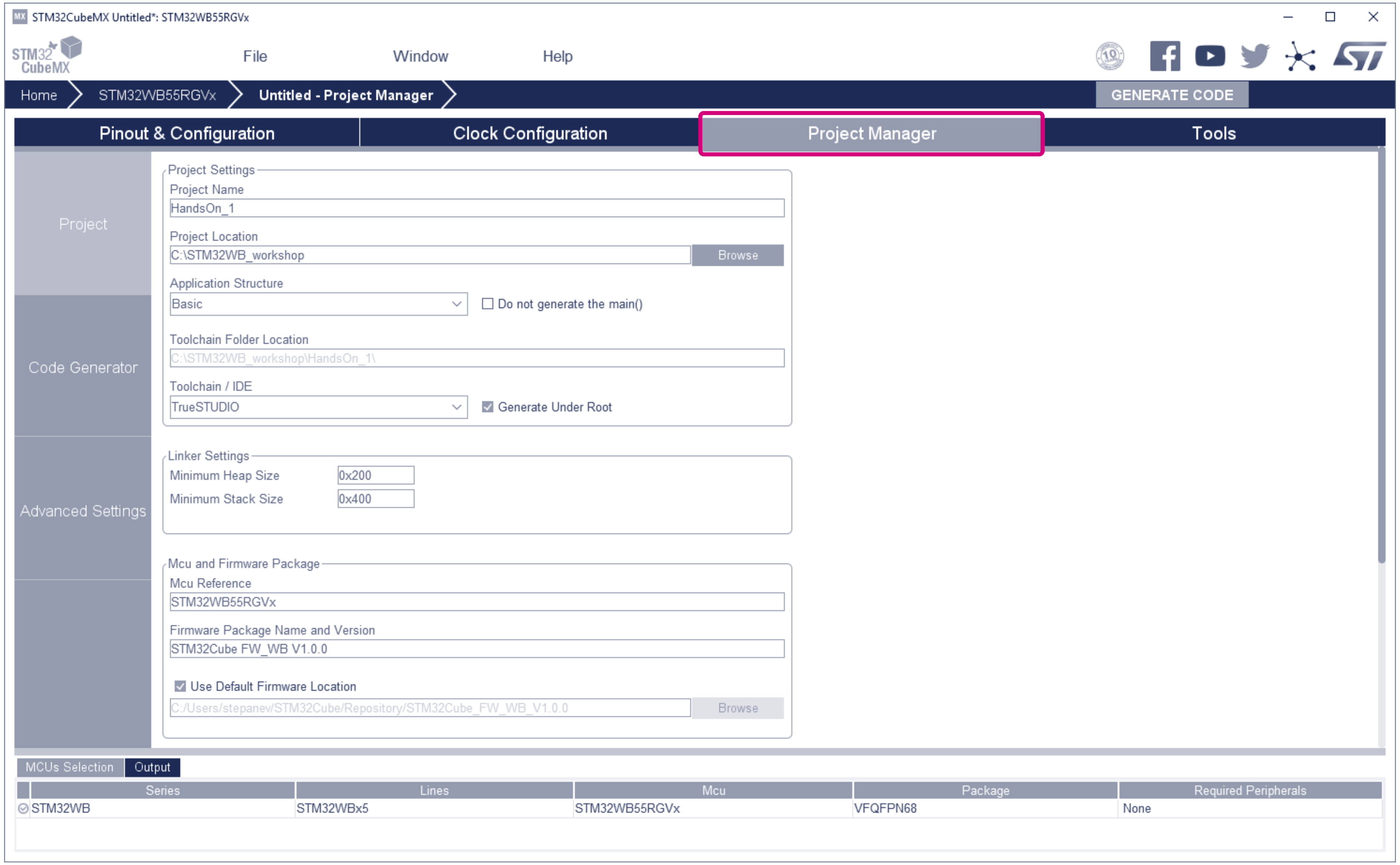
The clocks requirements for select peripherals (e.g. 32MHz for RF) are not fulfilled, click on Yes, when the Clock issue solver dialog pops up

Clock configuration



HSE 32MHz
SYSCLK HSE_SYS
→ HCLKx 32MHz
LSE 32.768kHz
RTC from LSE
RFWKP from LSE

Finalize the project settings



Project Settings

Project Name: HandsOn_1

Project Location: C:\STM32WB_workshop

Application Structure: Basic

☐ Do not generate the main()

Toolchain Folder Location: C:\STM32WB_workshop\HandsOn_1\

Toolchain / IDE: TrueSTUDIO

☑ Generate Under Root

Linker Settings

Minimum Heap Size	0x200
Minimum Stack Size	0x400

Mcu and Firmware Package

Mcu Reference: STM32WB55RGVx

Firmware Package Name and Version: STM32Cube FW_WB V1.0.0

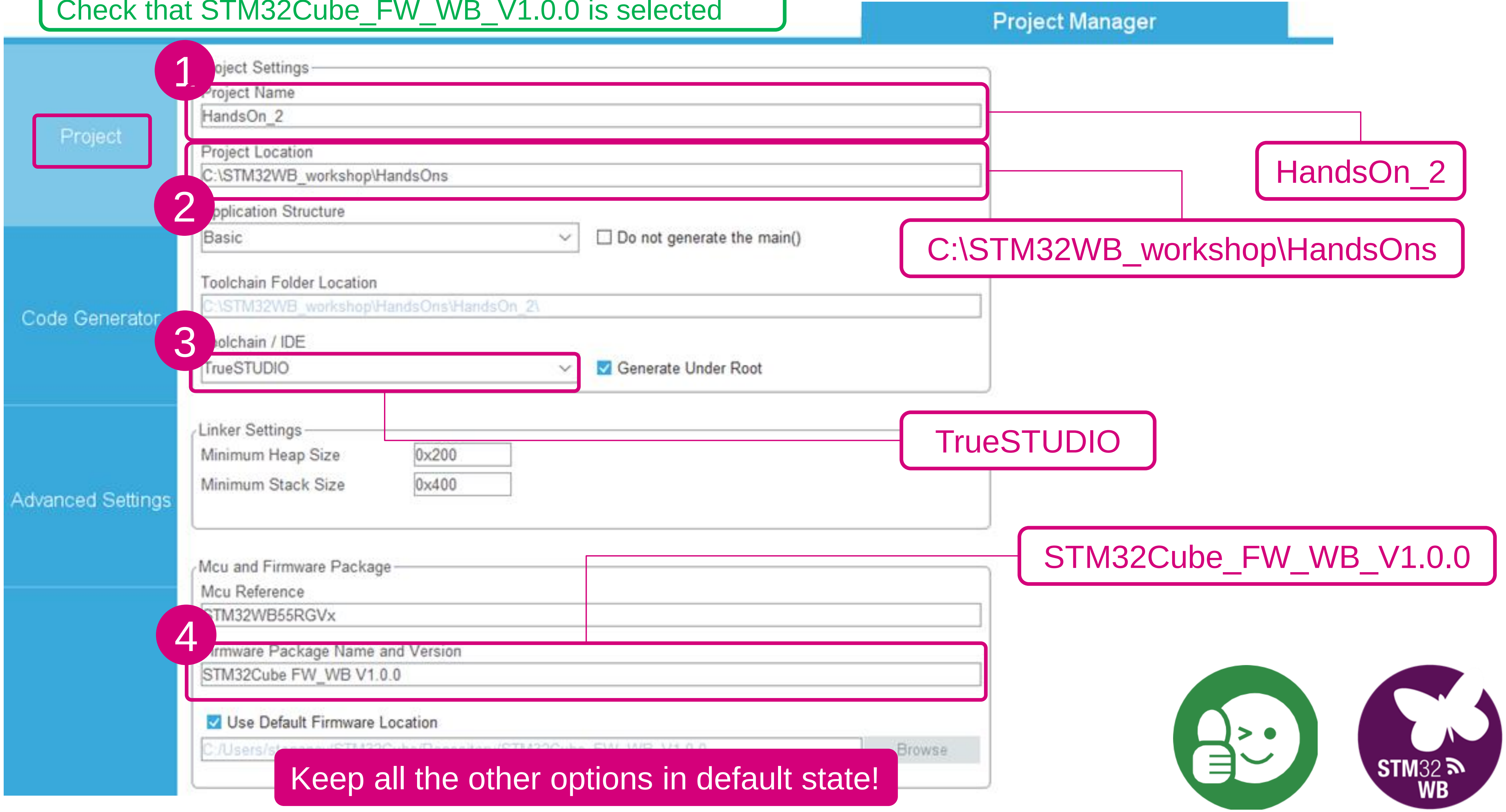
☑ Use Default Firmware Location

C:/Users/stepanev/STM32Cube/Repository/STM32Cube_FW_WB_V1.0.0

Series	Lines	Mcu	Package	Required Peripherals
STM32WB	STM32WBx5	STM32WB55RGVx	VFQFPN68	None

Project settings

Project Name → HandsOn_2
Project Location → C:\STM32WB_workshop\HandsOns\
IDE → TrueSTUDIO
Check that STM32Cube_FW_WB_V1.0.0 is selected



Project Manager

Project | Code Generator | Advanced Settings

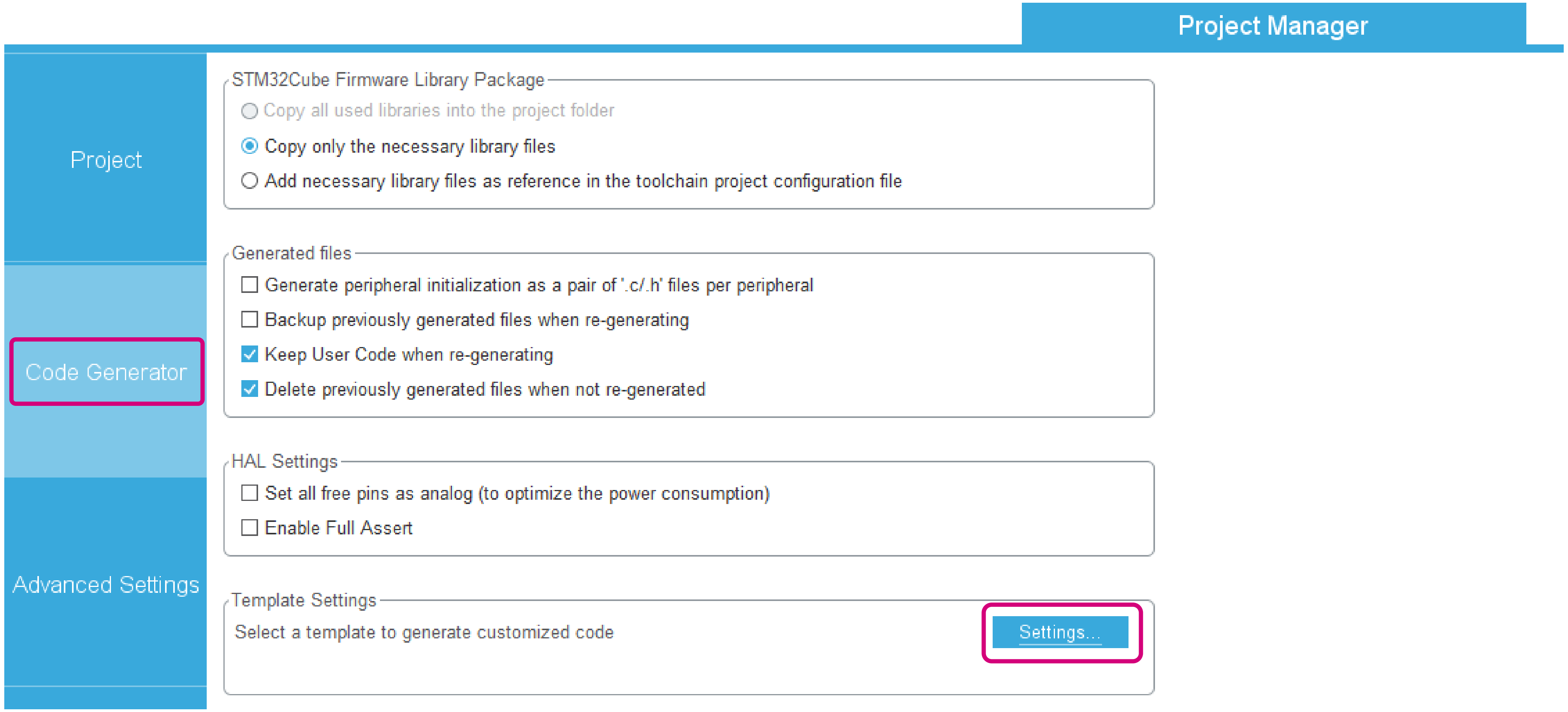
Project Settings
1. Project Name: HandsOn_2
2. Project Location: C:\STM32WB_workshop\HandsOns
Application Structure: Basic — ☐ Do not generate the main()
Toolchain Folder Location: C:\STM32WB_workshop\HandsOns\HandsOn_2\
3. Toolchain / IDE: TrueSTUDIO — ☑ Generate Under Root

Linker Settings
Minimum Heap Size: 0x200
Minimum Stack Size: 0x400

Mcu and Firmware Package
Mcu Reference: STM32WB55RGVx
4. Firmware Package Name and Version: STM32Cube FW_WB V1.0.0
☑ Use Default Firmware Location

Keep all the other options in default state!

Add the user templates for OLED driver



STM32Cube Firmware Library Package

- ○ Copy all used libraries into the project folder
- ● Copy only the necessary library files
- ○ Add necessary library files as reference in the toolchain project configuration file

Generated files

- ☐ Generate peripheral initialization as a pair of '.c/.h' files per peripheral
- ☐ Backup previously generated files when re-generating
- ✓ Keep User Code when re-generating
- ✓ Delete previously generated files when not re-generated

HAL Settings

- ☐ Set all free pins as analog (to optimize the power consumption)
- ☐ Enable Full Assert

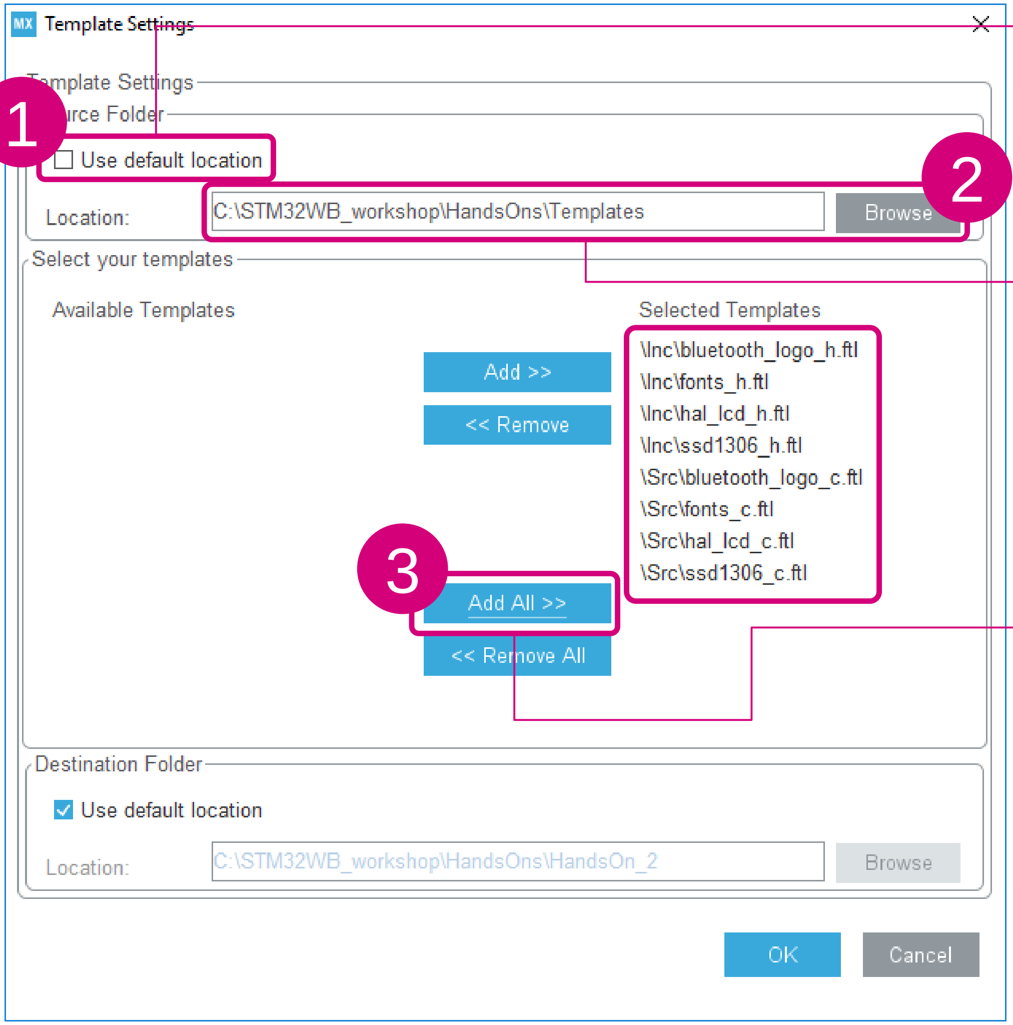
Template Settings

Select a template to generate customized code

Project Manager
→ Code Generator tab
→ Template Settings

Project Manager
→ Code Generator tab
→ Template Settings

User Templates settings



Template Settings

Template Settings

Source Folder

☐ Use default location

Location: C:\STM32WB_workshop\HandsOns\Templates Browse

Select your templates

Available Templates

Selected Templates

\Inc\bluetooth_logo_h.ftl
\Inc\fonts_h.ftl
\Inc\hal_lcd_h.ftl
\Inc\ssd1306_h.ftl
\Src\bluetooth_logo_c.ftl
\Src\fonts_c.ftl
\Src\hal_lcd_c.ftl
\Src\ssd1306_c.ftl

Add >>
<< Remove
Add All >>
<< Remove All

Destination Folder

☑ Use default location

Location: C:\STM32WB_workshop\HandsOns\HandsOn_2 Browse

OK Cancel

Uncheck "Use default location"

Select Templates folder inside the STM32WB_workshop materials: C:\STM32WB_workshop\HandsOns\Templates

Press "Add All >>" to add all 8 files to the project

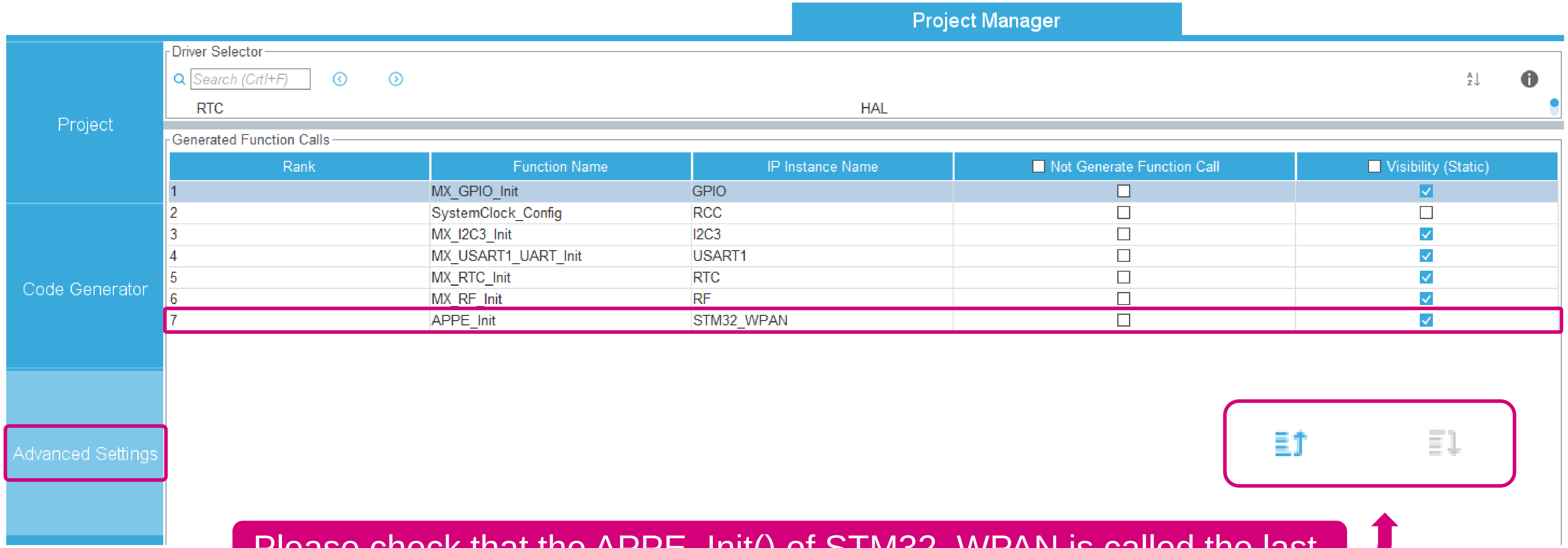
OLED driver files prepared for this workshop

bluetooth_logo_h.ftl, fonts_h.ftl, hal_lcd_h.ftl, ssd1306_h.ftl
bluetooth_logo_c.ftl, fonts_c.ftl, hal_lcd_c.ftl, ssd1306_c.ftl

life.augmented

STM32 WB

Check that the STM32_WPAN init is called last



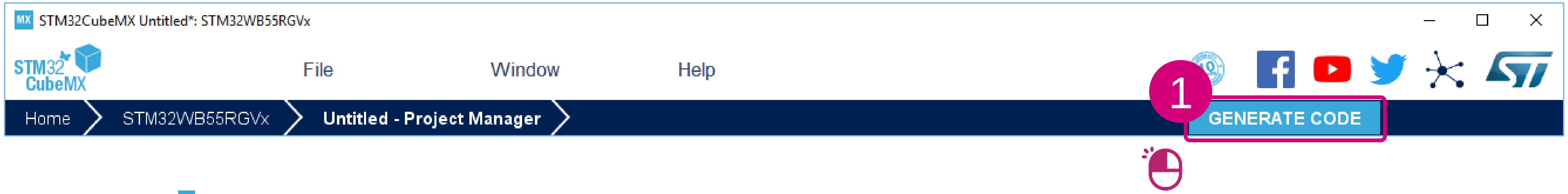
RTC	HAL

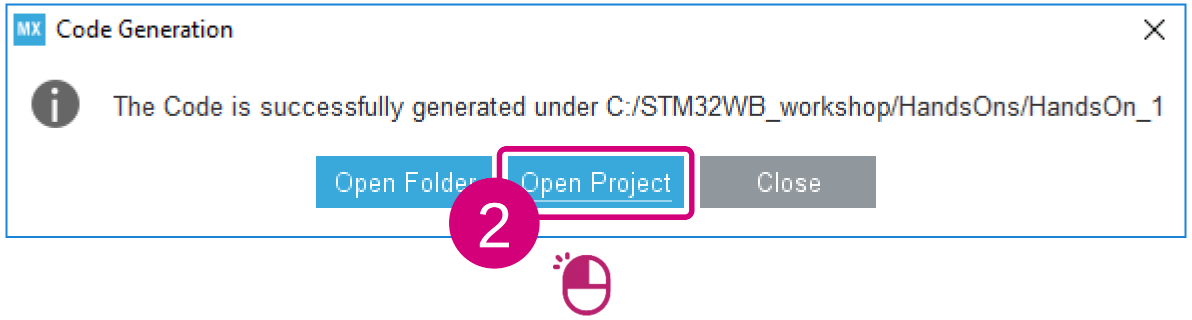
Generated Function Calls

Rank	Function Name	IP Instance Name	☐ Not Generate Function Call	☐ Visibility (Static)
1	MX_GPIO_Init	GPIO	☐	✓
2	SystemClock_Config	RCC	☐	☐
3	MX_I2C3_Init	I2C3	☐	✓
4	MX_USART1_UART_Init	USART1	☐	✓
5	MX_RTC_Init	RTC	☐	✓
6	MX_RF_Init	RF	☐	✓
7	APPE_Init	STM32_WPAN	☐	✓

Please check that the APPE_Init() of STM32_WPAN is called the last, if not, move it using the buttons at the bottom of the panel

Generate the code


STM32CubeMX Untitled*: STM32WB55RGVx
STM32 CubeMX
File
Window
Help
Home
STM32WB55RGVx
Untitled - Project Manager
1
GENERATE CODE

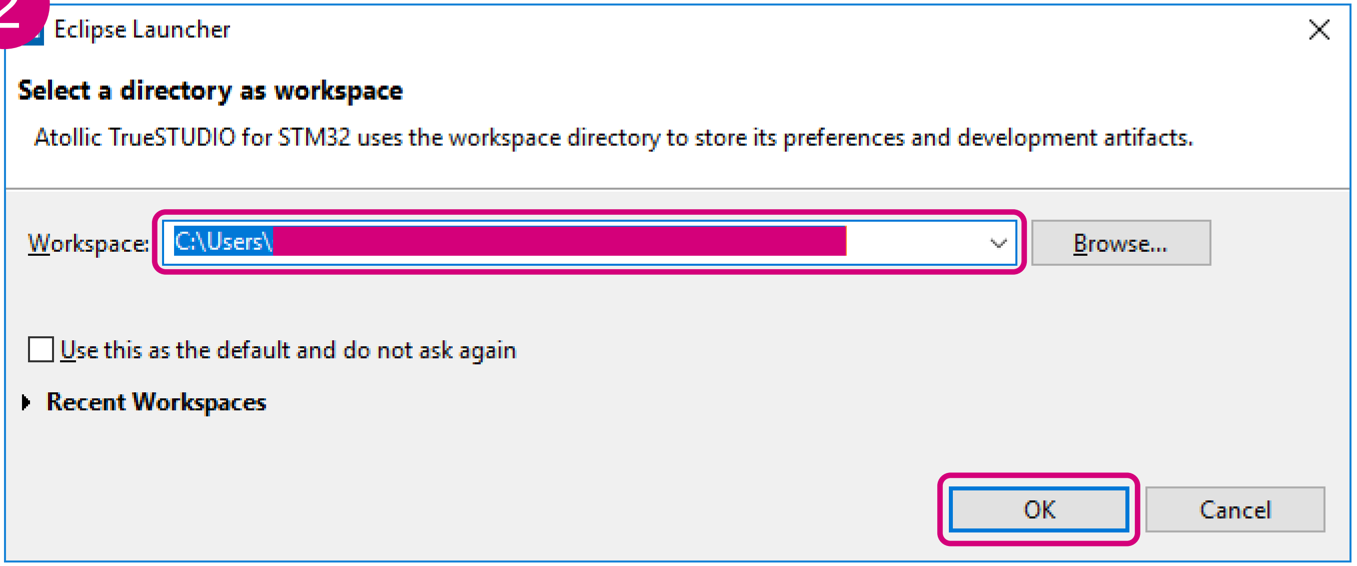


The Code is successfully generated under C:/STM32WB_workshop/HandsOns/HandsOn_1

Atollic TrueStudio project opening

1

2



Select a directory as workspace

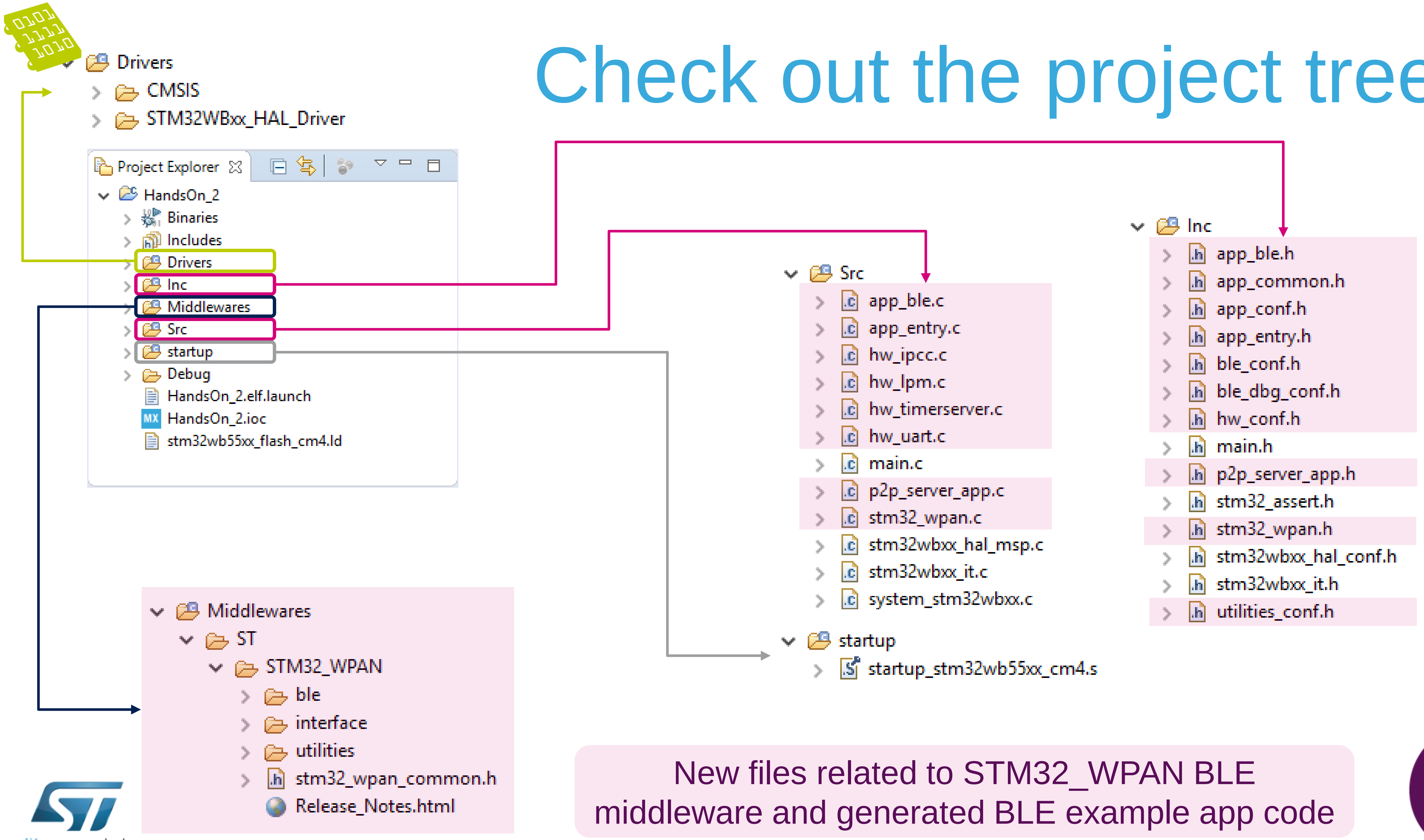
Atollic TrueSTUDIO for STM32 uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\[redacted]

☐ Use this as the default and do not ask again

Select the workspace

Check out the project tree



Drivers
- CMSIS
- STM32WBxx_HAL_Driver

Project Explorer
- HandsOn_2
  - Binaries
  - Includes
  - Drivers
  - Inc
  - Middlewares
  - Src
  - startup
  - Debug
  - HandsOn_2.elf.launch
  - HandsOn_2.ioc
  - stm32wb55xx_flash_cm4.ld

Middlewares
- ST
  - STM32_WPAN
    - ble
    - interface
    - utilities
    - stm32_wpan_common.h
    - Release_Notes.html

Src
- app_ble.c
- app_entry.c
- hw_ipcc.c
- hw_lpm.c
- hw_timerserver.c
- hw_uart.c
- main.c
- p2p_server_app.c
- stm32_wpan.c
- stm32wbxx_hal_msp.c
- stm32wbxx_it.c
- system_stm32wbxx.c

startup
- startup_stm32wb55xx_cm4.s

Inc
- app_ble.h
- app_common.h
- app_conf.h
- app_entry.h
- ble_conf.h
- ble_dbg_conf.h
- hw_conf.h
- main.h
- p2p_server_app.h
- stm32_assert.h
- stm32_wpan.h
- stm32wbxx_hal_conf.h
- stm32wbxx_it.h
- utilities_conf.h

New files related to STM32_WPAN BLE middleware and generated BLE example app code

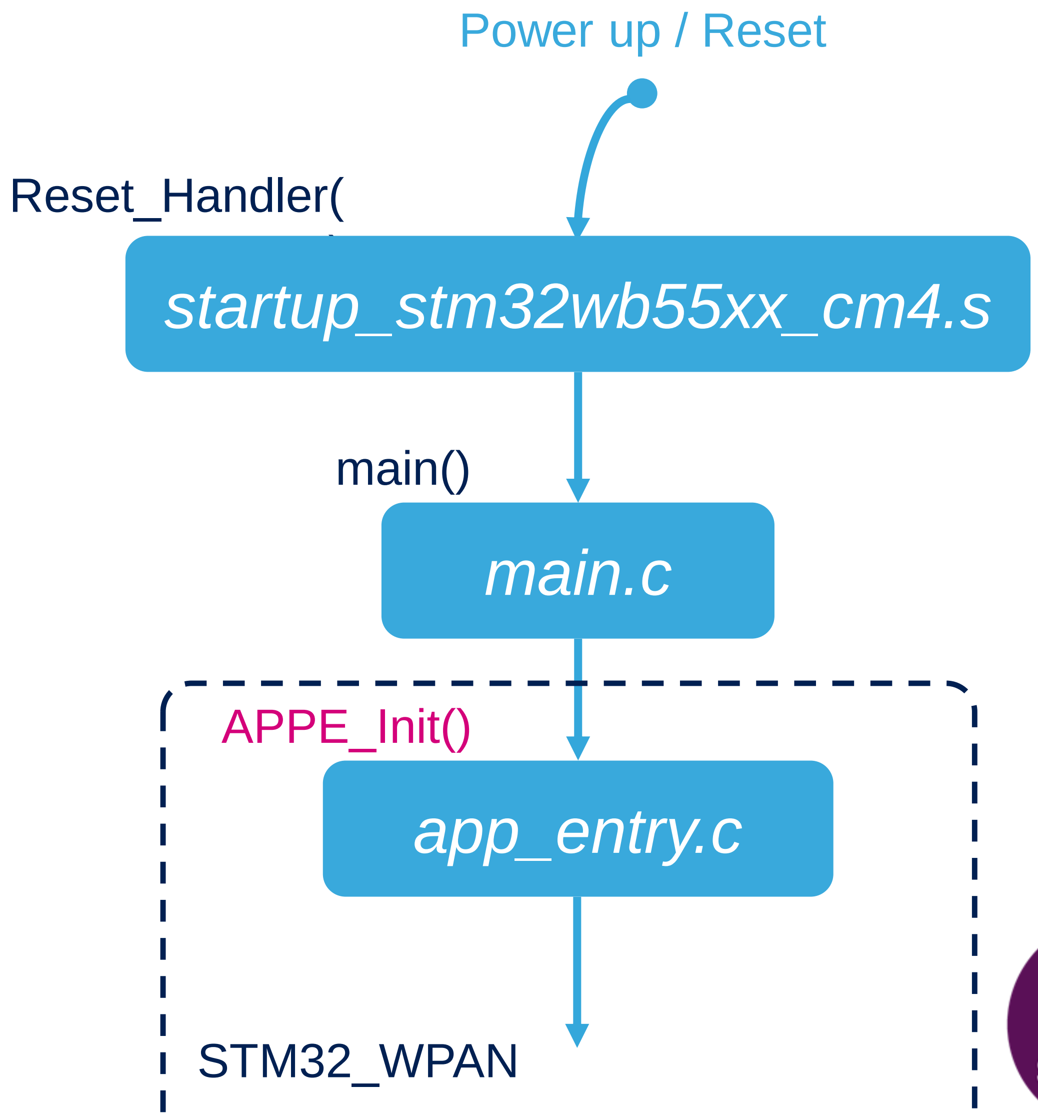
STM32 system blocks and IPs initialization

- Reset_Handler
  - stack pointer initialization
  - Variables initialization (SRAM memory)
  - SystemInit() call

    → main()

- main()
  - MCU HW initialization
    - RCC (clock), GPIO, RTC, I2C3,…

    → APPE_Init()



stm32wbxx_hal_msp.c

Add the HSE tuning

user section **{ Includes @Line ~24 }**

1

```
/* USER CODE BEGIN Includes */
#include "otp.h"
/* USER CODE END Includes */
```

ADD

STEP1_Add_HSE_tuning.txt

HAL_MspInit(…) user section **{ MspInit 0 @Line ~67}**

2

```
void HAL_MspInit(void)
{
/* USER CODE BEGIN MspInit 0 */
  #warning “Following code is valid only for P-NUCLEO-WB55 boards and should be re-
implemented depending on the target HW and HSE capacitor tuning value storage location.“
  OTP_ID0_t * p_otp;

  /**
    * Read HSE_Tuning from OTP
    */
  p_otp = (OTP_ID0_t *) OTP_Read(0);
  if (p_otp)
  {
    LL_RCC_HSE_SetCapacitorTuning(p_otp->hse_tuning);
  }
/* USER CODE END MspInit 0 */

}
```

New feature of STM32WB Target HW specific

AN5042

life.augmented

Add STM32_WPAN ISRs

stm32wbxx_it.c

ADD

user section **{ Includes @Line ~25 }**

1

```
/* USER CODE BEGIN Includes */
#include "app_common.h“
/* USER CODE END Includes */
```

user section **{ 1 @Line ~215 }**

2

```
/* USER CODE BEGIN 1 */
/**
  * @brief This function handles RTC wake-up interrupt through EXTI line 19.
  */
void RTC_WKUP_IRQHandler(void)
{
  HW_TS_RTC_Wakeup_Handler();
}
/**
  * @brief This function handles IPCC RX occupied interrupt.
  */
void IPCC_C1_RX_IRQHandler(void)
{
  HW_IPCC_Rx_Handler();
}
/**
  * @brief This function handles IPCC TX free interrupt.
  */
void IPCC_C1_TX_IRQHandler(void)
{
  HW_IPCC_Tx_Handler();
}
/* USER CODE END 1 */
```

STEP2_Add_STM32_WPAN_ISRs.txt

These interrupt handlers and callbacks implemented in STM32_WPAN modules are currently not generated by STM32CubeMX when STM32_WPAN is in use

IPCC managed by STM32_WPAN completely

life.augmented

STM32 WB

Typical simple application architecture

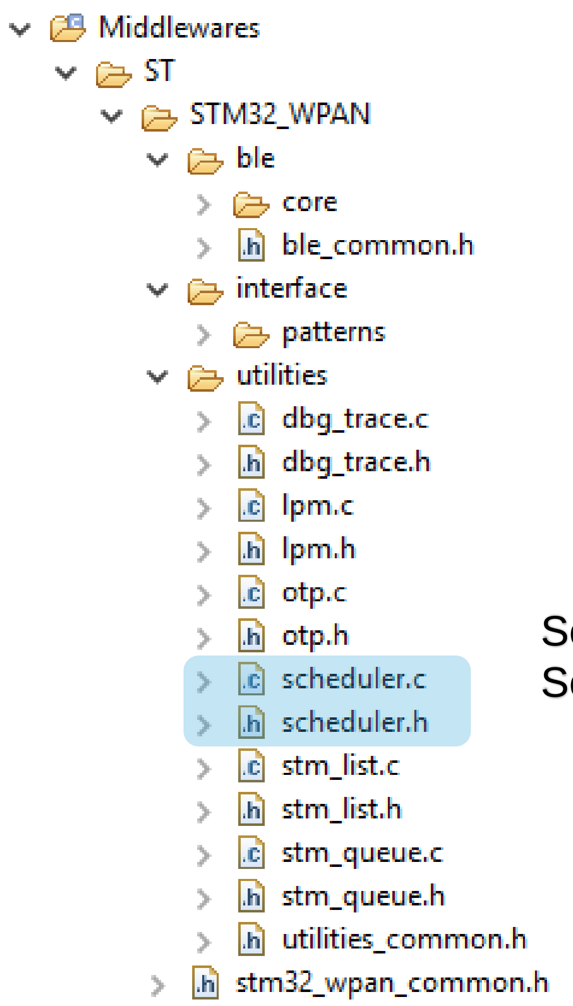
```
main() {
…
  while(1) {
    switch(event):
    case EVENT1:
      Task1();
      clear_EVENT1();
    …
    case EVENTX:
      TaskX();
      clear_EVENTX();
    case IDLE:
      Enter_Low_Power_Mode();
    default:
      break;
  }
}
```

Sequencer (Simple Scheduler) as a basic task manager

Simple Scheduler

The scheduler provides the following features:

- ✓ Up to 32 tasks registered
- ✓ Request a task to be executed
- ✓ Pause and Resume a task
- ✓ Wait for a specific event (might be not blocking)
- ✓ Priority on tasks



```
v Middlewares
  v ST
    v STM32_WPAN
      v ble
        > core
        > ble_common.h
      v interface
        > patterns
      v utilities
        > dbg_trace.c
        > dbg_trace.h
        > lpm.c
        > lpm.h
        > otp.c
        > otp.h
        > scheduler.c
        > scheduler.h
        > stm_list.c
        > stm_list.h
        > stm_queue.c
        > stm_queue.h
        > utilities_common.h
      > stm32_wpan_common.h
```

Scheduler.c
Scheduler.h

Simple Scheduler

- Register a task to be executed in the background at any time / any place in the firmware (before it is requested to be executed)
- Enter low power mode when there is nothing to schedule
- Request the scheduler to execute a task according to priority in the background. The request may be done at any time / any place in the firmware (from interrupt handler, function, etc…)
- List of API
  - SCH_Idle()
  - SCH_Run()
  - SCH_RegTask()
  - SCH_SetTask()

  - SCH_PauseTask()
  - SCH_ResumeTask()
  - SCH_WaitEvt()
  - SCH_SetEvt()
  - SCH_IsEvtPend()
  - SCH_EvtIdle()

```
Main( void )
{
  HAL_Init();
  .  .  .
 SCH_RegTask( Id1, Task1);
  SCH_RegTask( Id2, task2);
  .  .  .
while(1)
  {
    SCH_Run(~0);
  }
}

void SCH_Idle(  void )
{
  LPM_EnterModeSelected();
}

void fct  ( void )
{
 SCH_SetTask( Id1, Prio0);
}

void fct_IT  ( void )
{
 SCH_SetTask( Id2, Prio1);
}
```

Register tasks to be executed in the background

Enter low power mode when there is nothing to schedule

Request the scheduler to execute Task1 in the background

Request the scheduler to execute Task2 in the background

Add STM32_WPAN scheduler call

main.c

user section **{ Includes @Line ~40 }**

1

```
/* USER CODE BEGIN Includes */
#include "scheduler.h"
/* USER CODE END Includes */
```

ADD

user section **{ 3 @Line ~116 }**

2

```
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  SCH_Run(~0);
}
/* USER CODE END 3 */
```

Not generated by STM32CubeMX yet when STM32_WPAN is in use.

STEP3_Add_scheduler_call.txt

life.augmented

STM32 WB

OLED display functionality

Display the Device name

STEP4_Add_OLED_init.txt

app_ble.c

user section **{ Includes @Line ~40 }**

ADD

1

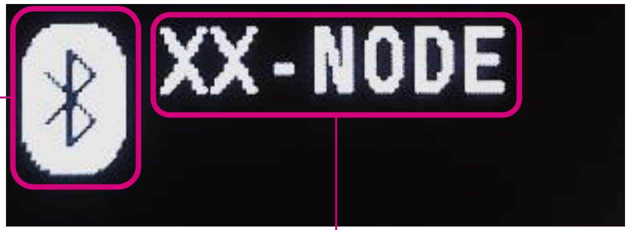
```
/* USER CODE BEGIN Includes */
#include "hal_lcd.h"
/* USER CODE END Includes */
```

user section **{ APP_BLE_Init_1 @Line ~381 }**

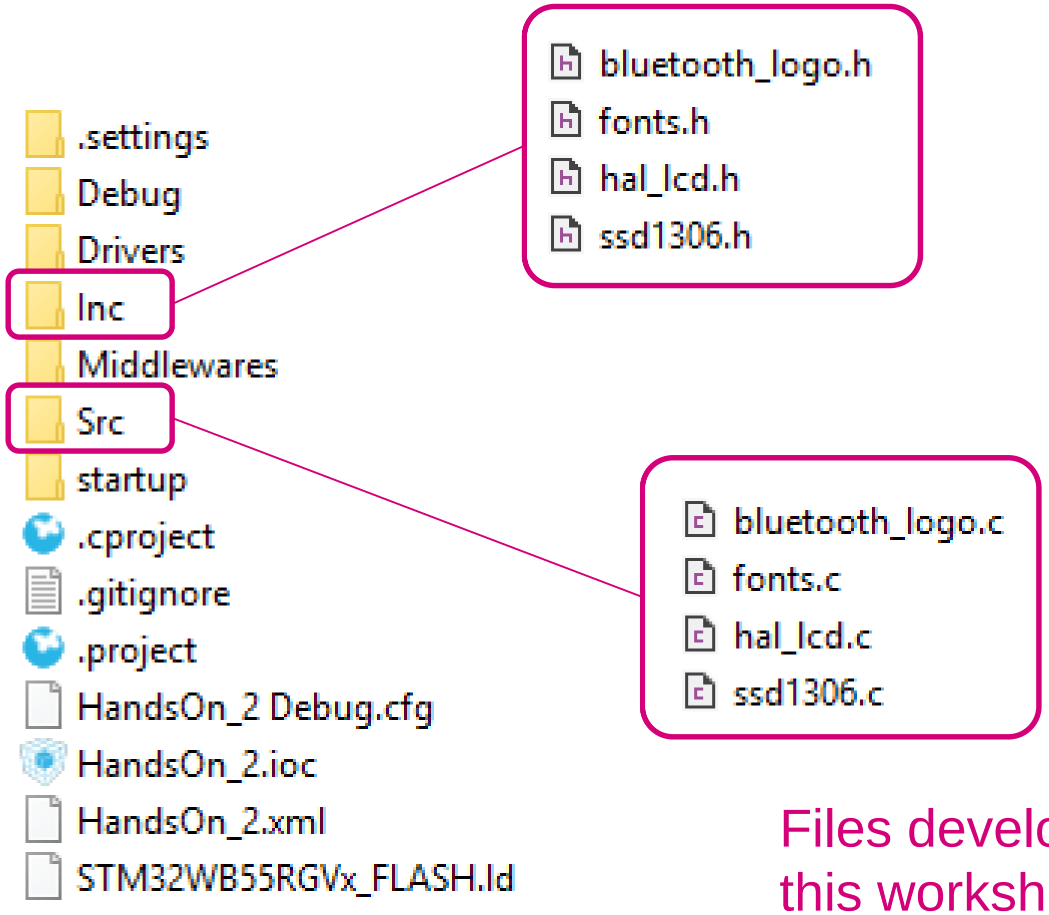
2

```
/* USER CODE BEGIN APP_BLE_Init_1 */
  /* Initialize the LCD */
  LCD_Init();
  /* Display the application icon */
  LCD_BLE_PrintLogo();
  /* Display the local device name */
  LCD_BLE_PrintLocalName(local_name);
/* USER CODE END APP_BLE_Init_1 */
```

According to the STM32_WPAN architecture, LCD_Init() shall be called from app_entry.c. We will put in app_ble.c just for simplicity.



OLED display driver files



.settings
Debug
Drivers
Inc
Middlewares
Src
startup
.cproject
.gitignore
.project
HandsOn_2 Debug.cfg
HandsOn_2.ioc
HandsOn_2.xml
STM32WB55RGVx_FLASH.ld

bluetooth_logo.h
fonts.h
hal_lcd.h
ssd1306.h

bluetooth_logo.c
fonts.c
hal_lcd.c
ssd1306.c

Files developed for this workshop only

Add the green LED blinking

app_ble.c : Line ~631

STEP5_Add_GREEN_LED_blinking.txt

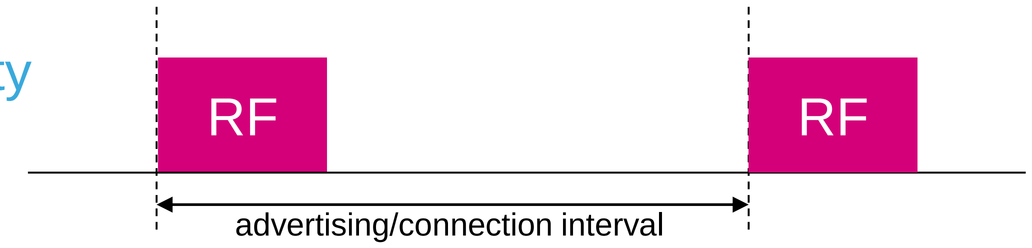
ADD

SVCCTL_App_Notification(…) user section **{ RADIO_ACTIVITY_EVENT }**

```
/* USER CODE BEGIN RADIO_ACTIVITY_EVENT */
HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_SET);
HAL_Delay(5);
HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_RESET);
/* USER CODE END RADIO_ACTIVITY_EVENT */
```

Generate 5ms flash with BLUE LED.

RADIO_ACTIVITY_EVENT
Triggered after every Radio RF activity finishes
Event mask configurable
(ADVERTISE, SCAN, CONNECTION)



RADIO_ACTIVITY_EVENT enabled in app_conf.h

1 Build

or Ctrl+B

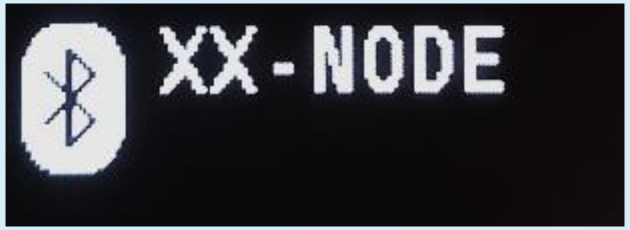
2 Debug

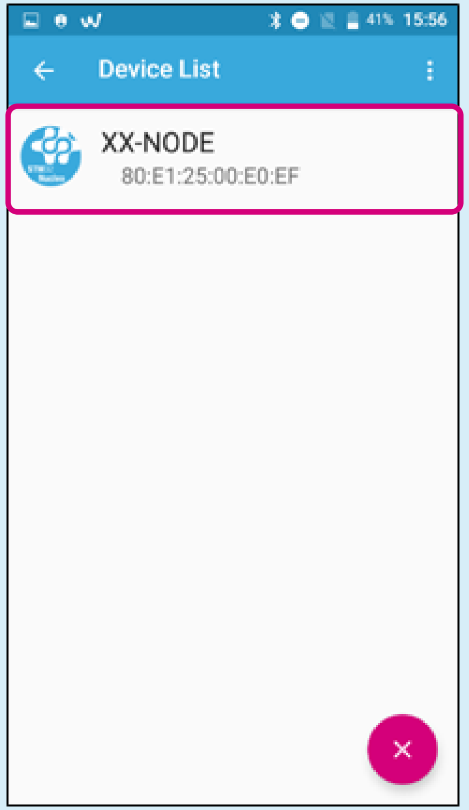
F11

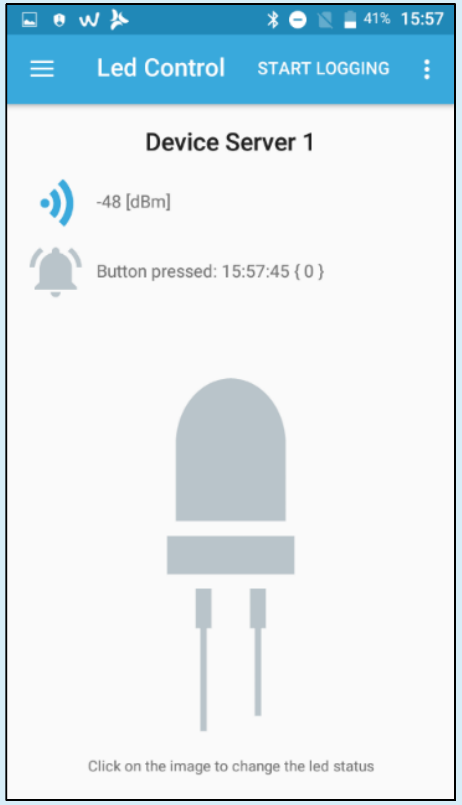
3 Resume

F8

Test the functionality






41% 15:57
Led Control
START LOGGING
Device Server 1
-48 [dBm]
Button pressed: 15:57:45 { 0 }
Click on the image to change the led status

GREEN LED blinking period changes when advertising vs. connected

Advertising stops after 60sec if link not established (GREEN LED stops blinking)

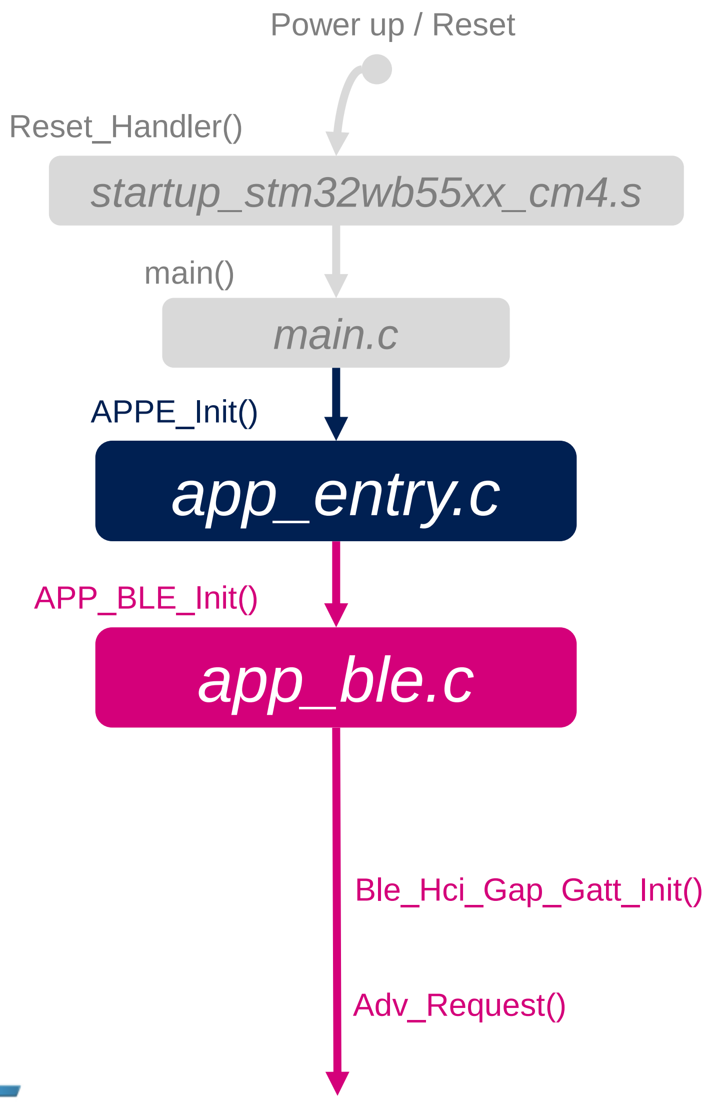
Reset for restart

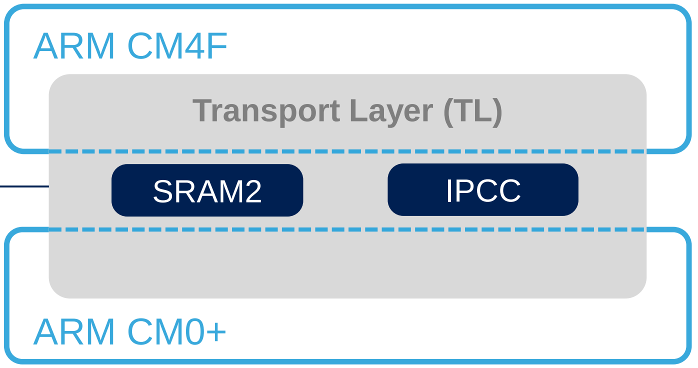
DISCONNECT

life.augmented

STM32 WB

BLE Application Initialization





Initialize **TL** (boot up CM0+) + other app specific HW

→ Wait for initialization done given boot up sequence

Initialize the BLE stack

(**GATT** and **GAP** initialization), advertising mode control

→ Set TX Output Power

→ GATT init

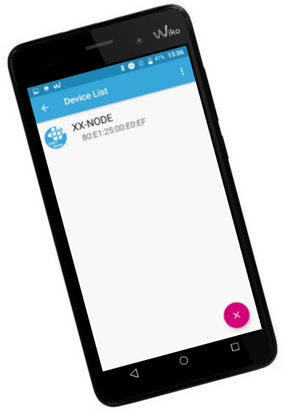
→ GAP init

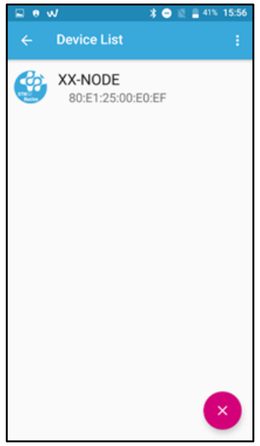
→ **Set discoverable (start advertising)**

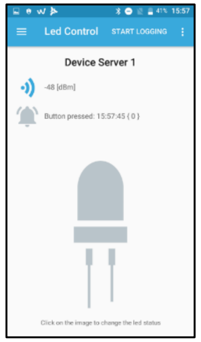
Generated code

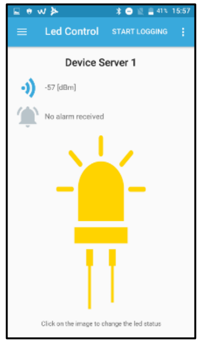
What is running now?

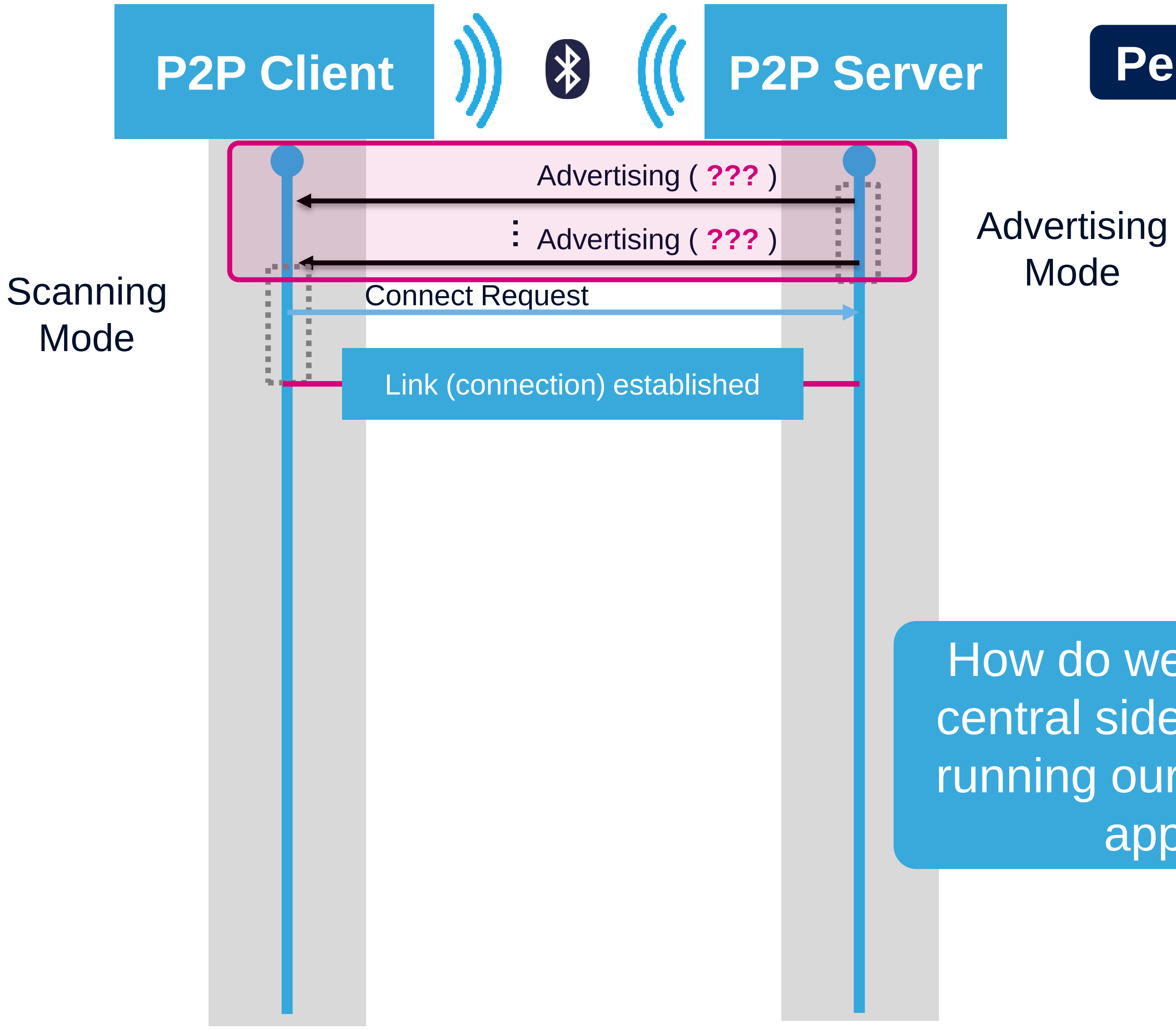
Central **GATT Client**



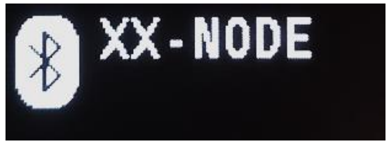



Led Control
START LOGGING
Device Server 1
-48 [dBm]
Button pressed: 15:57:45 (0)
Click on the image to change the led status





Peripheral **GATT Server**



How do we filter out on the central side just the devices running our simple example application?

BLE Advertising data

Over-The-Air BLE Packet

Length	1 byte	4 bytes	2~257 bytes	3 bytes
Name	Preamble	Access Address	Protocol Data Unit (PDU)	CRC
Value	10101010b	0xXXXXXXXX	0xXX……XX	0xXXXXXX

Don't care now

Don't care now

Advertising PDU

Length	2 bytes	6 bytes	0~31 bytes
Name	Header	Advertising Address	Advertising Data
Value	0xXXXX	0xXXXXXXXXXXXX	0xXX……XX

Don't care now

Interesting for us

Data PDU

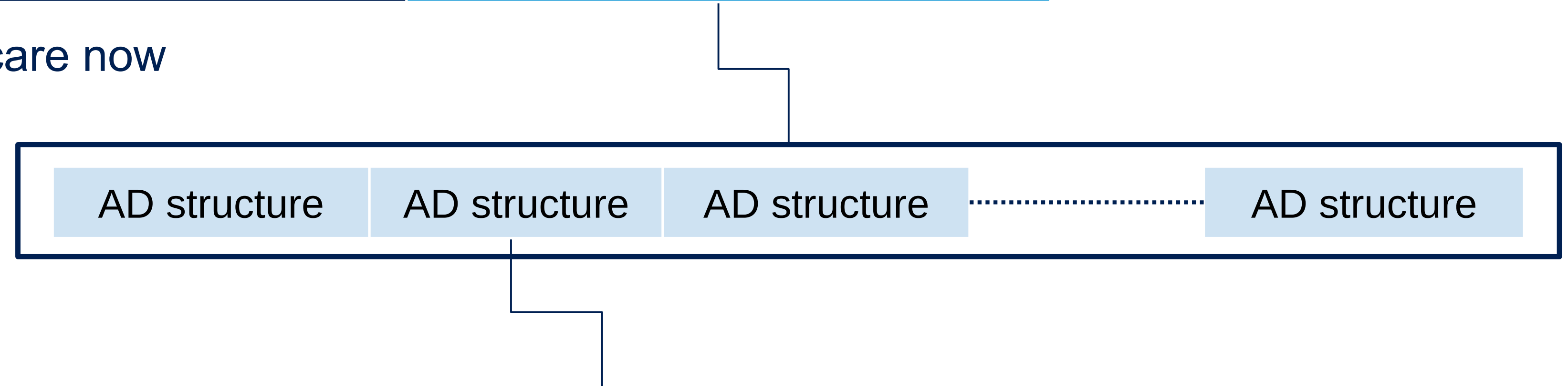
Not under scope now

BLE Advertising data

Advertising PDU

Length	2 bytes	6 bytes	0~31 bytes
Name	Header	Advertising Address	Advertising Data
Value	0xXXXX	0xXXXXXXXXXXXX	0xXX…………………………………………XX

Don’t care now



AD structure format

Length	1 byte	1 byte	(Length – 1) bytes
Name	Length	Type	Data

Several types defined, e.g.:
0x09 – Complete local name
0xFF – Manufacturer specific data

Advertised complete local name

AD structure of complete local name

Length	1 byte	1 byte	(Length – 1) bytes
Name	Length	Type	Complete local name
Value	0xXX	0x09	0xXXXX…………………XX

e.g. { 'X', 'X', '-', 'N', 'O', 'D', 'E' }

app_ble.c

Private variables

```
static const char local_name[] = { AD_TYPE_COMPLETE_LOCAL_NAME,'X','X','-','N','O','D','E' };
```

Advertising start

app_ble.c

Adv_Request(…) { } called at the end of **APP_BLE_Init**(…) { }

```
static void Adv_Request( void )
{
        ⋮
    ret = aci_gap_set_discoverable(
          ADV_IND,
          Min_Interval,
          Max_Interval,
          PUBLIC_ADDR,
          NO_WHITE_LIST_USE, /* use white list */
          sizeof(local_name),
          (uint8_t*) local_name,
          BleApplicationContext.BleApplicationContext_legacy.advtServUUIDlen,
          BleApplicationContext.BleApplicationContext_legacy.advtServUUID,
          0,
          0);
        ⋮
}
```

Start advertising ⟶ *aci_gap_set_discoverable(…);*

BlueST Protocol

AD structure of our BlueST Protocol

Length	1 byte	1 byte	1 byte	1 byte	4 bytes	6 bytes
Name	Length	Type	Protocol Version	Device Id	Feature Mask	Device MAC (optional)
Value	0x07/0xD	0xFF	0x01	0xXX	0xXXXXXXXX	0xXXXXXXXXXXXX

16-bit Company ID provided by Bluetooth SIG should be used here normally

0x00 for a generic device
0x01 - STEVAL-WESU1 board
0x02 - STEVAL-STLKT01V1 (SensorTile) board
0x03 - STEVAL-BCNKT01V1 (BlueCoin) board
0x04 - STEVAL-IDB008V1/2 (BlueNRG-2) board
0x05 - STEVAL-BCN002V1B (BlueNRG-Tile) board
0x80 to 0x8A for various functional packs for Nucleo boards

BlueST Protocol description (@github)

life.augmented

STM32 WB

app_ble.c

Manufacturer specific data

user code section **{ PV }**

```
/* USER CODE BEGIN PV */
 ⋮

/* Manufacturer specific data */
uint8_t manuf_data[14] = { sizeof(manuf_data)-1, /* AD_RECORD Length */
                           AD_TYPE_MANUFACTURER_SPECIFIC_DATA, /* AD_RECORD Type */
                           0x01, /* Protocol Version */
                           CFG_DEV_ID_P2P_SERVER1, /* Device Id */
                           0x00, /* GROUP A Feature  */
                           0x00, /* GROUP A Feature */
                           0x00, /* GROUP B Feature */
                           0x00, /* GROUP B Feature */
                           0x00, /* BLE MAC start -MSB */
                           0x00,
                           0x00,
                           0x00,
                           0x00,
                           0x00  /* BLE MAC stop */
                           };
 ⋮
/* USER CODE END PV */
```

BlueST Protocol version to 0x01
1st byte of manufacturer specific data
(to identify BlueST protocol)

The Device ID is 0x83
2nd byte of manufacturer specific data
P2P Server 1
(according to BlueST protocol)

life.augmented

STM32 WB

Advertising data update

app_ble.c

Adv_Request(…) { } called at the end of **APP_BLE_Init**(…) { }

```
static void Adv_Request( void )
{
   ┆
   ┆

   /* Update Advertising data */
   ret = aci_gap_update_adv_data(sizeof(manuf_data), (uint8_t*) manuf_data);

   ┆
   ┆
}
```

Start advertising $\longrightarrow$ *aci_gap_set_discoverable(…);*

Update advertising data $\longrightarrow$ *aci_gap_update_adv_data(…);*

Advertising stop

app_ble.c

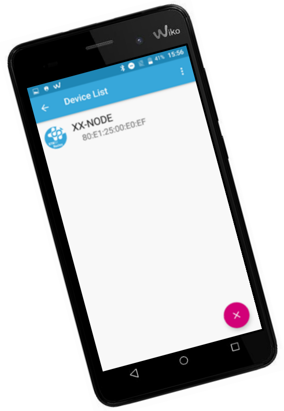
Adv_Cancel(…) { } called after defined timeout (60 sec – fully up to the user) { }

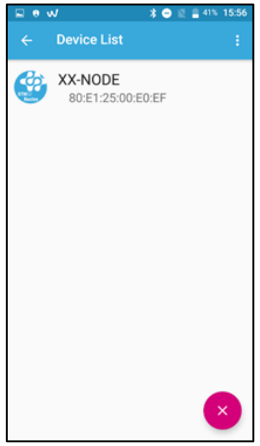
```
static void Adv_Cancel( void )
{
    ⋮
   result = aci_gap_set_non_discoverable();
    ⋮
}
```

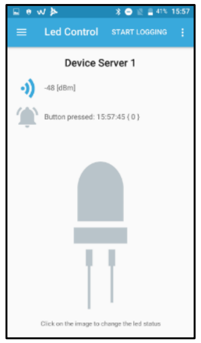
Start advertising ⟶ *aci_gap_set_discoverable(…);*
Update advertising data ⟶ *aci_gap_update_adv_data(…);*
Stop advertising ⟶ *aci_gap_set_non_discoverable(…);*

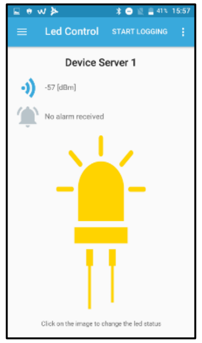
What is running now?

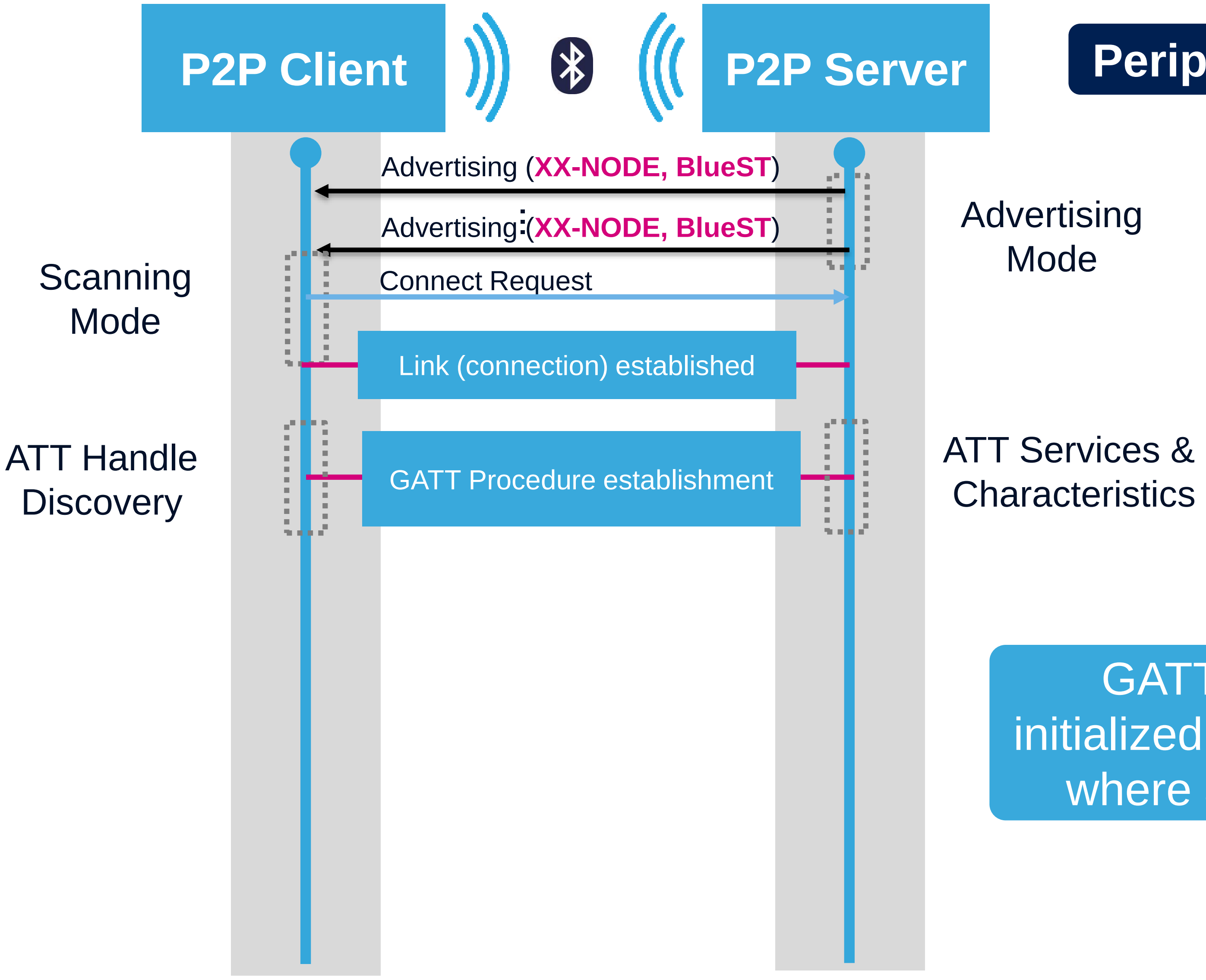
Central **GATT Client**



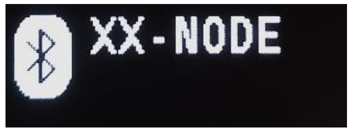



Led Control
START LOGGING
Device Server 1
-48 [dBm]
Button pressed: 15:57:45 (0)
Click on the image to change the led status




P2P Client
P2P Server
Advertising (XX-NODE, BlueST)
Advertising:(XX-NODE, BlueST)
Connect Request
Link (connection) established
GATT Procedure establishment
Scanning Mode
ATT Handle Discovery
Advertising Mode
ATT Services & Characteristics

Peripheral **GATT Server**

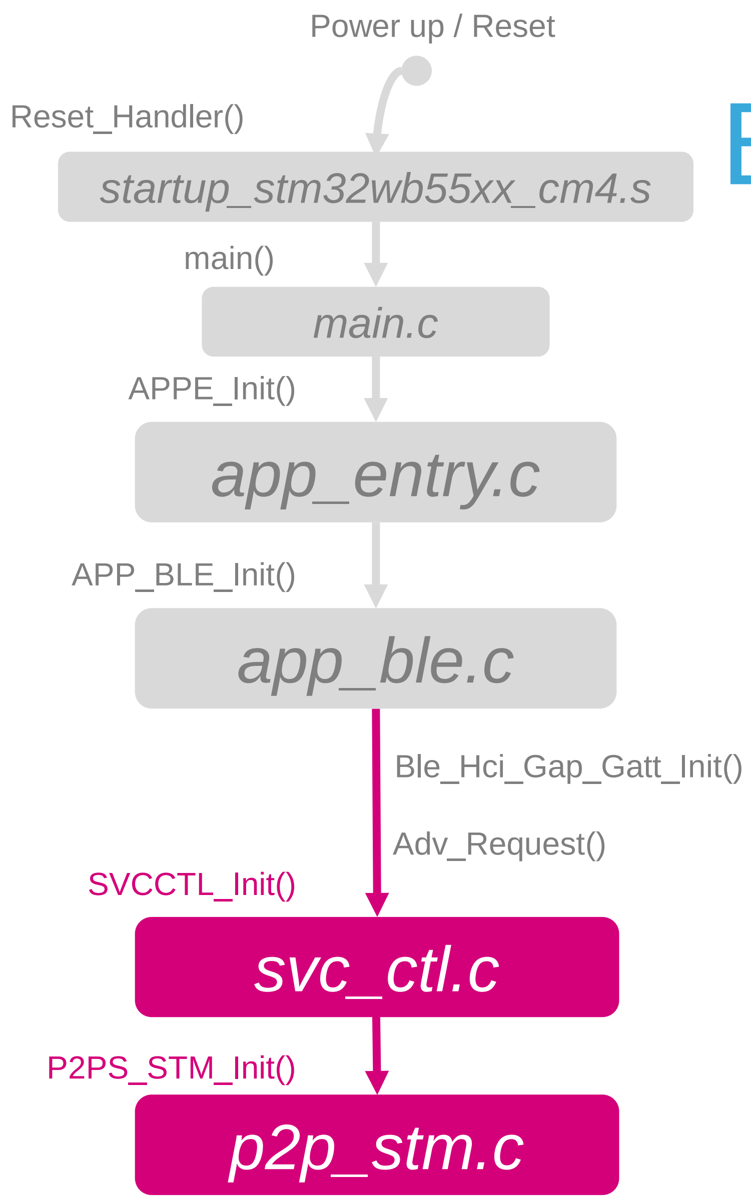


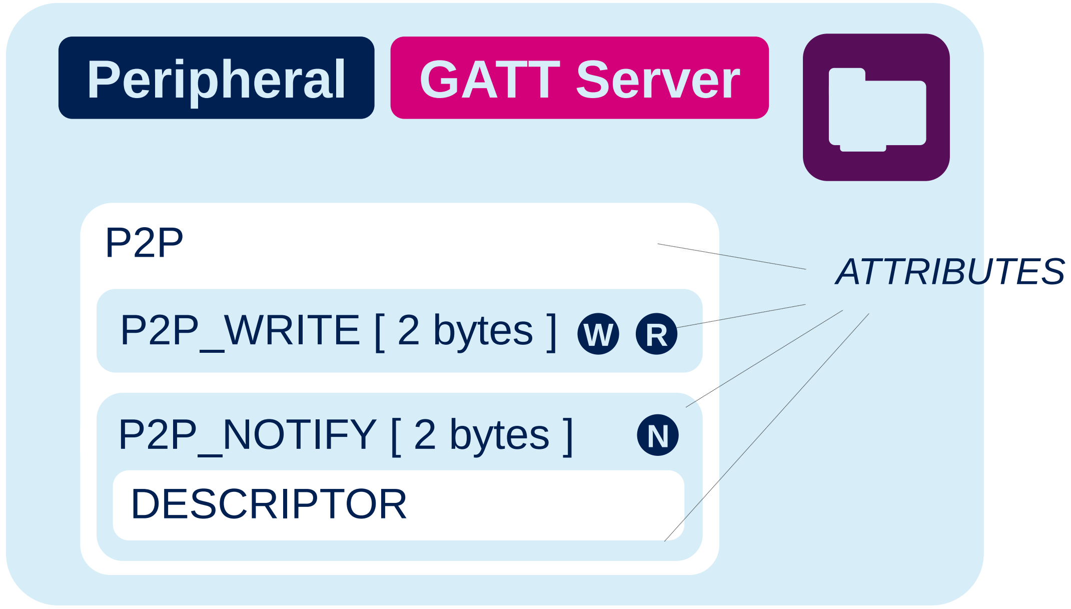
GATT Server initialized already too, where and how?

life.augmented

STM32 WB

BLE Application Initialization





Add **GATT** services and characteristics

Generated code

Add P2P STM service and characteristics

Add Service → *`aci_gatt_add_service(…)`*

Add Characteristic → *`aci_gatt_add_char(…)`*

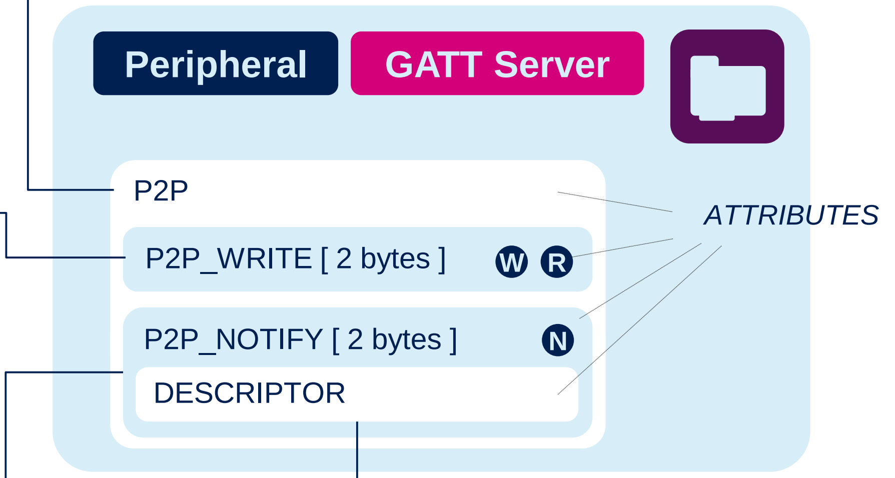
P2P_STM Service overview

UUID (hex)	`0000FE40-CC7A-482A-984A-7F2ED5B3E58F` (proprietary)
Type	PRIMARY SERVICE

UUID	`0000FE41-8E22-4541-9D4C-21EDAE82ED19` (proprietary)		
Properties	WRITE NO RESPONSE | READ		
Value	Byte	1 (LED state)	0 (Device number)
		0x00 – LED on 0x01 – LED off	0x00 - all 0x01~0x06 – P2P Server 1~6

UUID (hex)	`0000FE42-8E22-4541-9D4C-21EDAE82ED19` (proprietary)		
Properties	NOTIFY		
Value	Byte	1 (Button status)	0 (Device number)
		0x00 – button pressed 0x01 – button released	0x01~0x06 – P2P Server 1~6

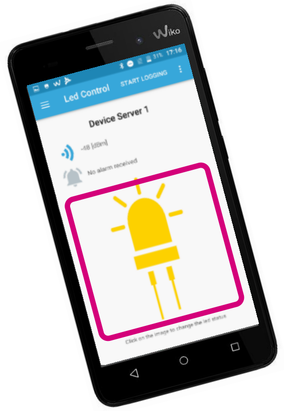
UUID (hex)	2902 (Client Characteristic Configuration)			
Value	Bit	15~2	1 (Indication state)	0 (Notification state)
		Reserved for future use	0 – Indications disabled 1 – Indications enabled	0 – Notifications disabled 1 – Notifications enabled

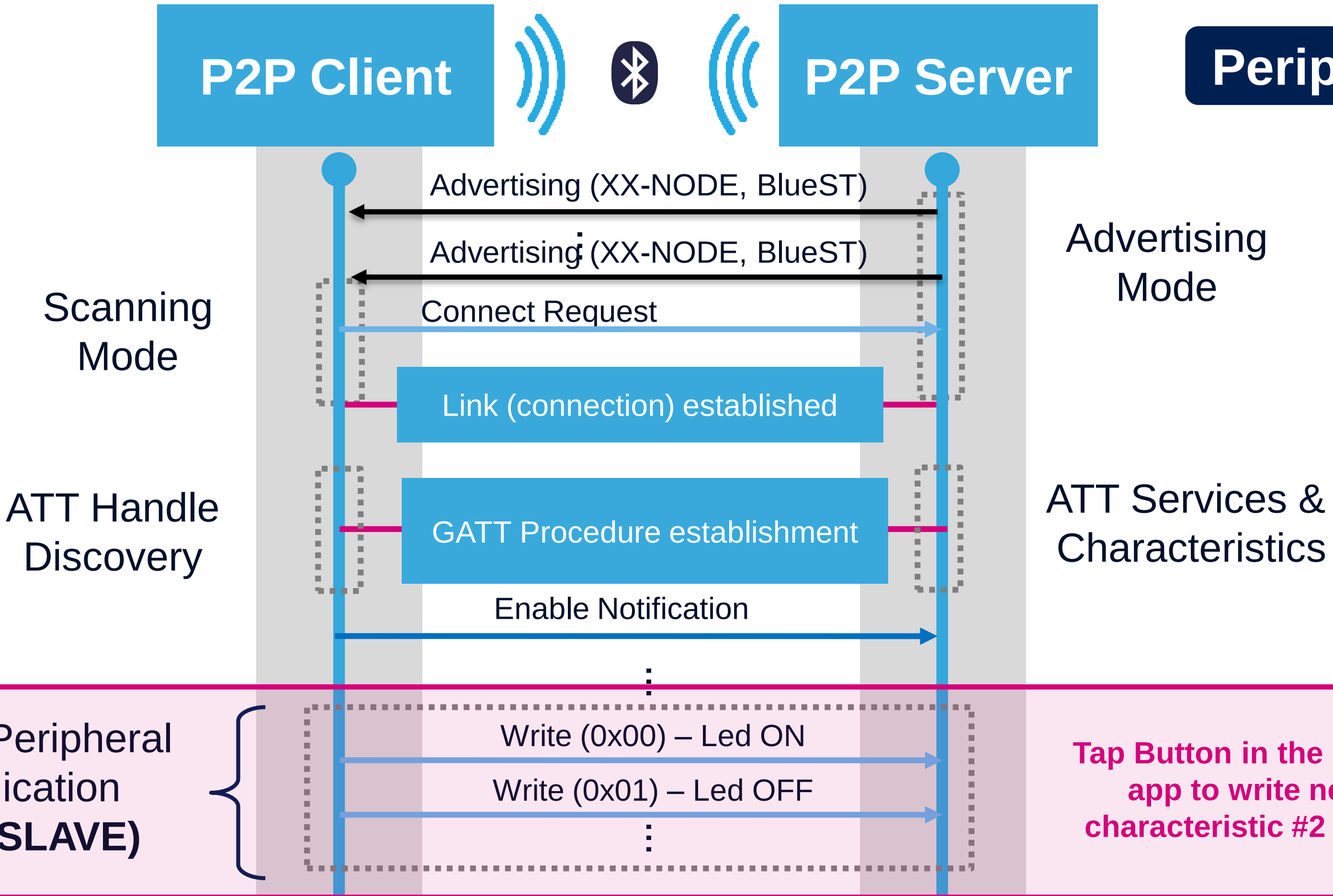


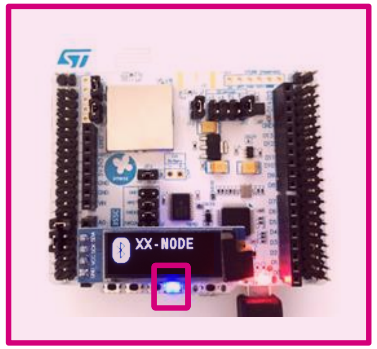
What to do **next?**

Central **GATT Client**

Peripheral **GATT Server**







Central-to-Peripheral communication **(Control SLAVE)**

Tap Button in the phone app to write new characteristic #2 value

Characteristic already being written, but the event and value not processed

Control the blue LED upon write characteristic event

P2P Server app GATT event handler

p2p_server_app.c

```
void P2PS_STM_App_Notification(P2PS_STM_App_Notification_evt_t *pNotification)
{
  switch(pNotification->P2P_Evt_Opcode)
  {
    case P2PS_STM__NOTIFY_ENABLED_EVT:
      
      break;
    case P2PS_STM_NOTIFY_DISABLED_EVT:
      
      break;
    case P2PS_STM_WRITE_EVT:
      /* Characteristic updated, parse the payload */
      
      break;
    default:
    break;
  }
  return;
}
```

EVT_BLUE_GATT_ATTRIBUTE_MODIFIED GATT events propagated from ***PeerToPeer_Event_Handler(…)*** registered @SVCCTL during ***P2PS_STM_Init()***

P2P_WRITE characteristic value changed

Attribute modified by client $\longrightarrow$ *EVT_BLUE_GATT_ATTRIBUTE_MODIFIED*

Add the blue LED control

STEP6_Add_BLUE_LED_control.txt

if 2nd byte of P2P_WRITE characteristic value is 0x01 → Turn the blue LED ON

p2p_server_app.c : Line ~85

P2PS_STM_App_Notification(…) user section **{ P2PS_STM_WRITE_EVT }**

ADD

```
void P2PS_STM_App_Notification(P2PS_STM_App_Notification_evt_t *pNotification)
{
    ⋮
/* USER CODE BEGIN P2PS_STM_WRITE_EVT */
    if(pNotification->DataTransfered.pPayload[1] == 0x01) {
      HAL_GPIO_WritePin(LED_BLUE_GPIO_Port, LED_BLUE_Pin, GPIO_PIN_SET);
    }
    else {
      HAL_GPIO_WritePin(LED_BLUE_GPIO_Port, LED_BLUE_Pin, GPIO_PIN_RESET);
    }
/* USER CODE END P2PS_STM_WRITE_EVT */
    ⋮
}
```

Turn the blue LED OFF otherwise

1st byte of the value as “don’t care” now

Test the functionality

1 Build

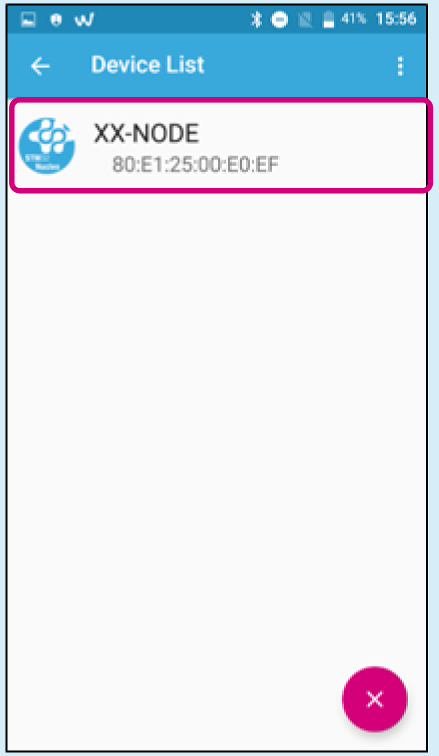
or Ctrl+B

2 Debug

F11

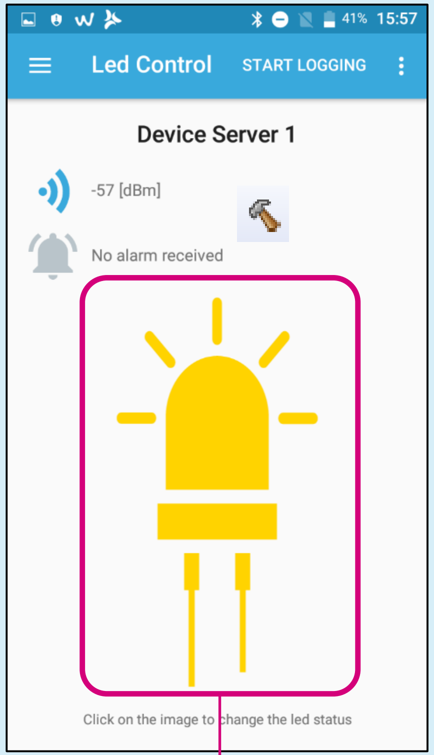
3 Resume

F8


Device List

XX-NODE

80:E1:25:00:E0:EF


Led Control

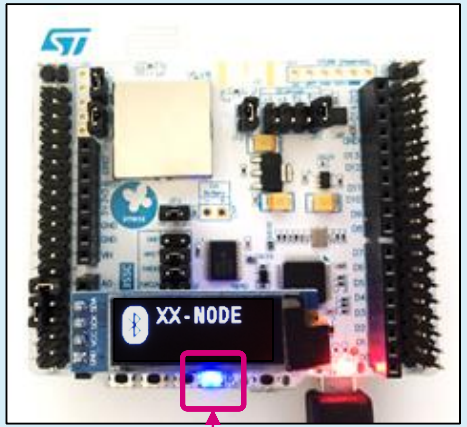
START LOGGING

Device Server 1

-57 [dBm]

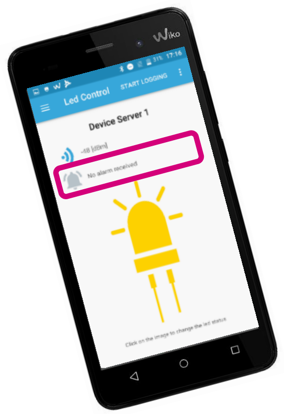
No alarm received

Click on the image to change the led status

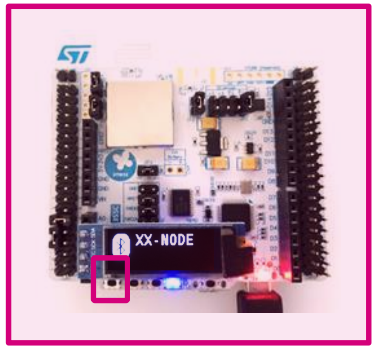


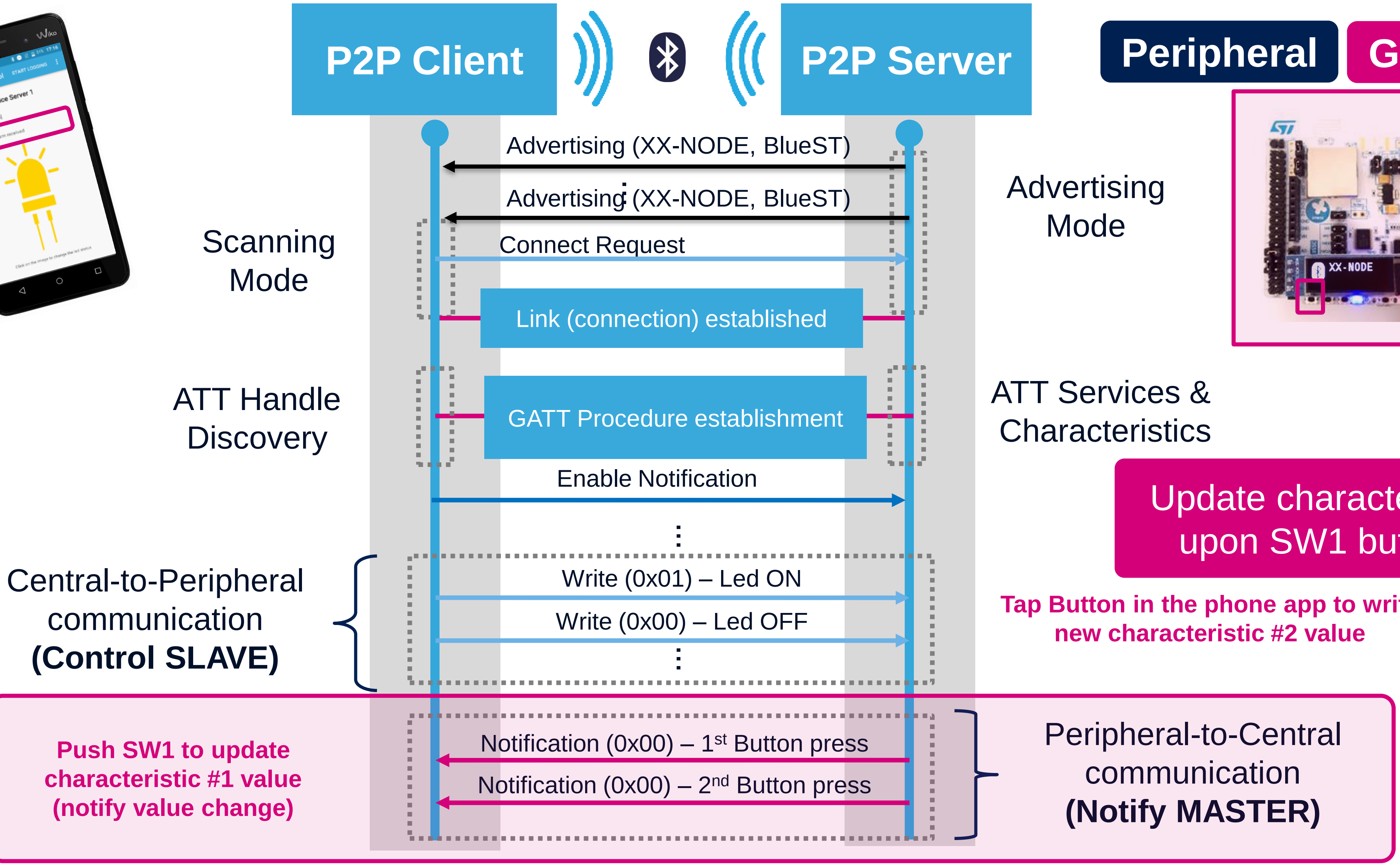
What to do **next?**

Central **GATT Client**



Peripheral **GATT Server**




P2P Client
P2P Server
Advertising (XX-NODE, BlueST)
Advertising (XX-NODE, BlueST)
Advertising Mode
Scanning Mode
Connect Request
Link (connection) established
ATT Handle Discovery
GATT Procedure establishment
ATT Services & Characteristics
Enable Notification
Central-to-Peripheral communication (Control SLAVE)
Write (0x01) – Led ON
Write (0x00) – Led OFF
Push SW1 to update characteristic #1 value (notify value change)
Notification (0x00) – 1st Button press
Notification (0x00) – 2nd Button press
Peripheral-to-Central communication (Notify MASTER)

Update characteristic value upon SW1 button press

Tap Button in the phone app to write new characteristic #2 value

Implement Send Notification function

p2p_server_app.c : Line ~55 user section **{ PFP }**

STEP7_Add_P2PS_Send_Notification.txt

ADD

1

```
/* USER CODE BEGIN PFP */
static void P2PS_Send_Notification(void);
/* USER CODE END PFP */
```

p2p_server_app.c : Line ~150 user section **{ FD }**

2

```
/* USER CODE BEGIN FD */
static void P2PS_Send_Notification(void)
{
  /* Update P2P_NOTIFY characteristic */
  P2PS_STM_App_Update_Char(P2P_NOTIFY_CHAR_UUID, 0x00);

  return;
}
/* USER CODE END FD */
```

p2p_stm.c

Update Characteristic value $\longrightarrow$ *aci_gatt_update_char_value(…)*

life.augmented

STM32 WB

Register Send Notification function as a task

p2p_server_app.c : Line ~141

STEP8_Add_P2PS_notify_task.txt

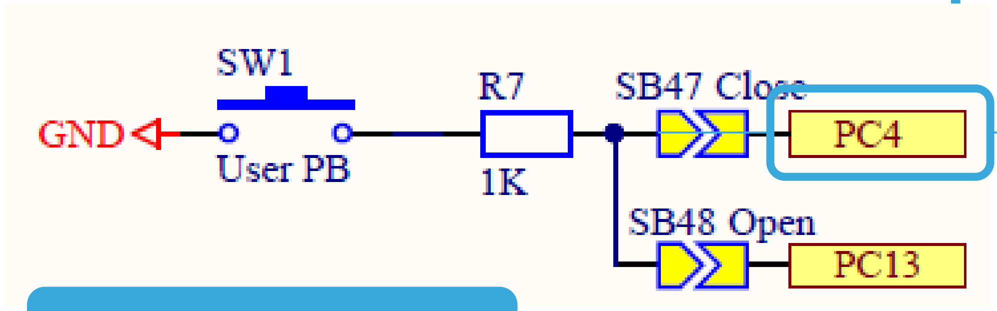
P2PS_APP_Init(…) user section **{ P2PS_APP_Init }**

```
void P2PS_APP_Init(void)
{
/* USER CODE BEGIN P2PS_APP_Init */
  SCH_RegTask( CFG_TASK_SW1_BUTTON_PUSHED_ID, P2PS_Send_Notification );

/* USER CODE END P2PS_APP_Init */
}
```

ADD

From where we should trigger this task now?

EXTI on GPIO pin connected to button SW1



PC4 pin already configured as following:

- GPIO_EXTIx
- BUTTON_SW1 label
- Internal Pull-Up enabled
- Falling-Edge detection activated
- EXTI Line 4 interrupt enabled in NVIC

stm32wbxx_it.c

```
void EXTI4_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI4_IRQn 0 */

  /* USER CODE END EXTI4_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_4);
  /* USER CODE BEGIN EXTI4_IRQn 1 */

  /* USER CODE END EXTI4_IRQn 1 */
}
```

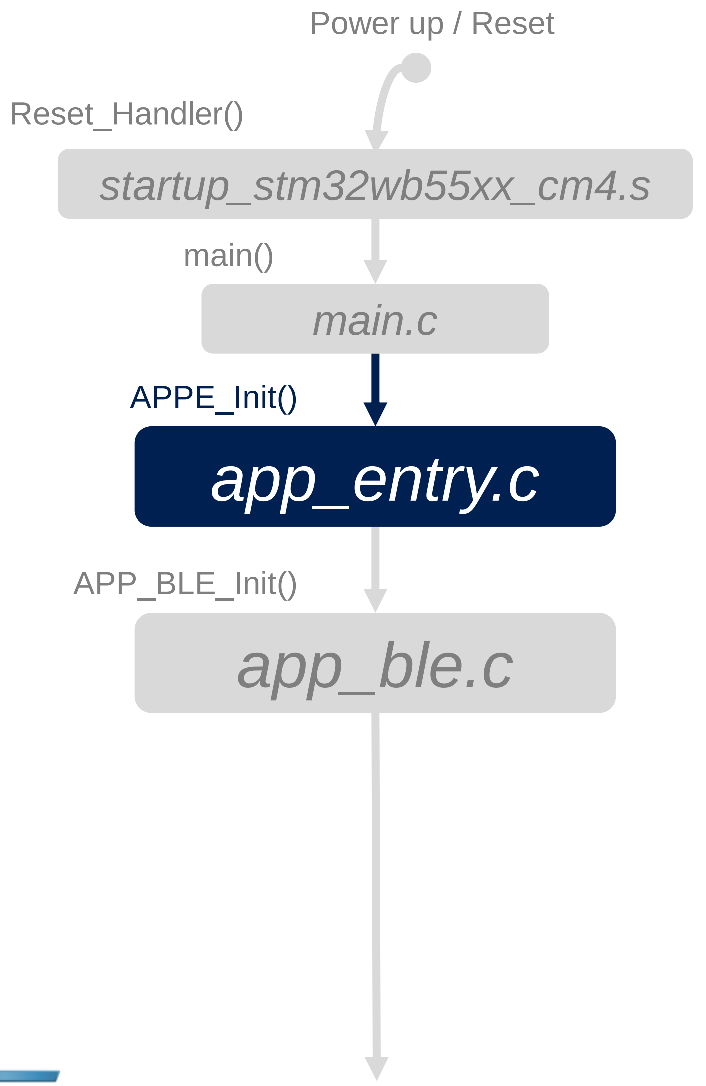
GENERATED CODE

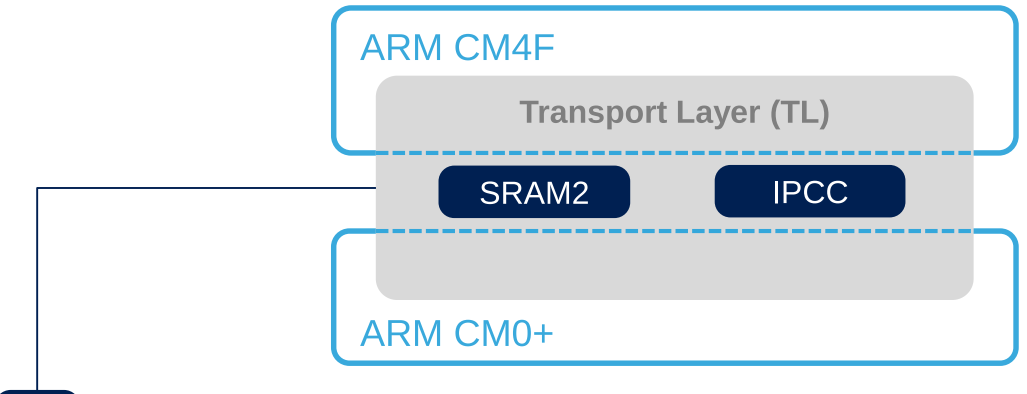
stm32wbxx_hal_gpio.c

```
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
  /* EXTI line interrupt detected */
  if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != 0x00u)
  {
    __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
    HAL_GPIO_EXTI_Callback(GPIO_Pin);
  }
```

TO BE IMPLEMENTED IN END USER CODE

Implement GPIO EXTI callback, where???





Initialize **TL** (boot up CM0+) + other app specific HW
→ Wait for initialization done **given boot up sequence**

We need to place somewhere HAL_GPIO_EXTI_IRQHandler(…) callback.

app_entry.c file is designed by the middleware architecture to hold the additional HW related functions. Also, in case you would like to use other EXTI channels for other purposes, the HAL callback function can be only one in the complete application code.

Call Send Notification from SW1 Button EXTI callback

app_entry.c : Line ~107

STEP9_Add_EXTI_callback.txt

ADD

user code section **{ FD }**

```
/* USER CODE BEGIN FD */

void HAL_GPIO_EXTI_Callback( uint16_t GPIO_Pin )
{
  switch (GPIO_Pin)
  {
  case BUTTON_SW1_Pin:
    SCH_SetTask(1<<CFG_TASK_SW1_BUTTON_PUSHED_ID, CFG_SCH_PRIO_0);
    break;
  default:
    break;
  }
  return;
}

/* USER CODE END FD */
```

Callback called from HAL_GPIO_EXTI_IRQHandler(…) implemented in stm32wb_hal_gpio.c and called from EXTI4_IRQHandler() in stm32wbxx_it.c

We are still in ISR !

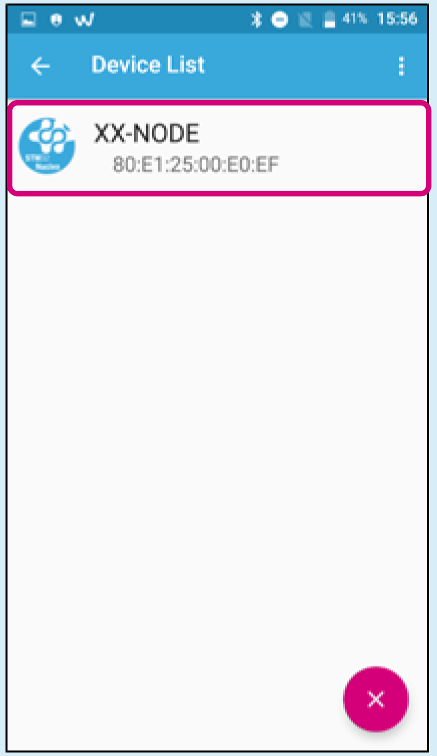
Proper Non-Blocking implementation

Test the functionality

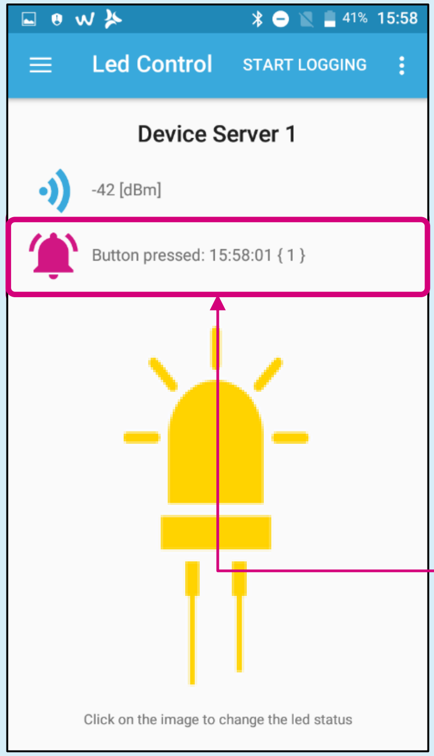
1 Build
or Ctrl+B

2 Debug
F11

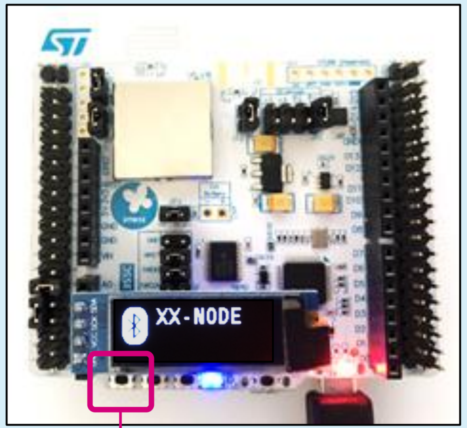
3 Resume
F8



XX-NODE
80:E1:25:00:E0:EF



Device Server 1
-42 [dBm]
Button pressed: 15:58:01 { 1 }
Click on the image to change the led status



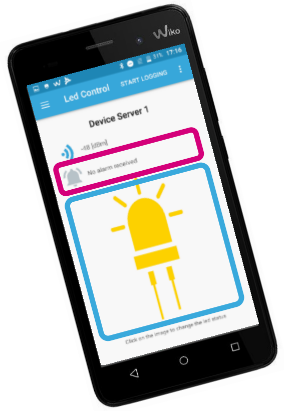
life.augmented

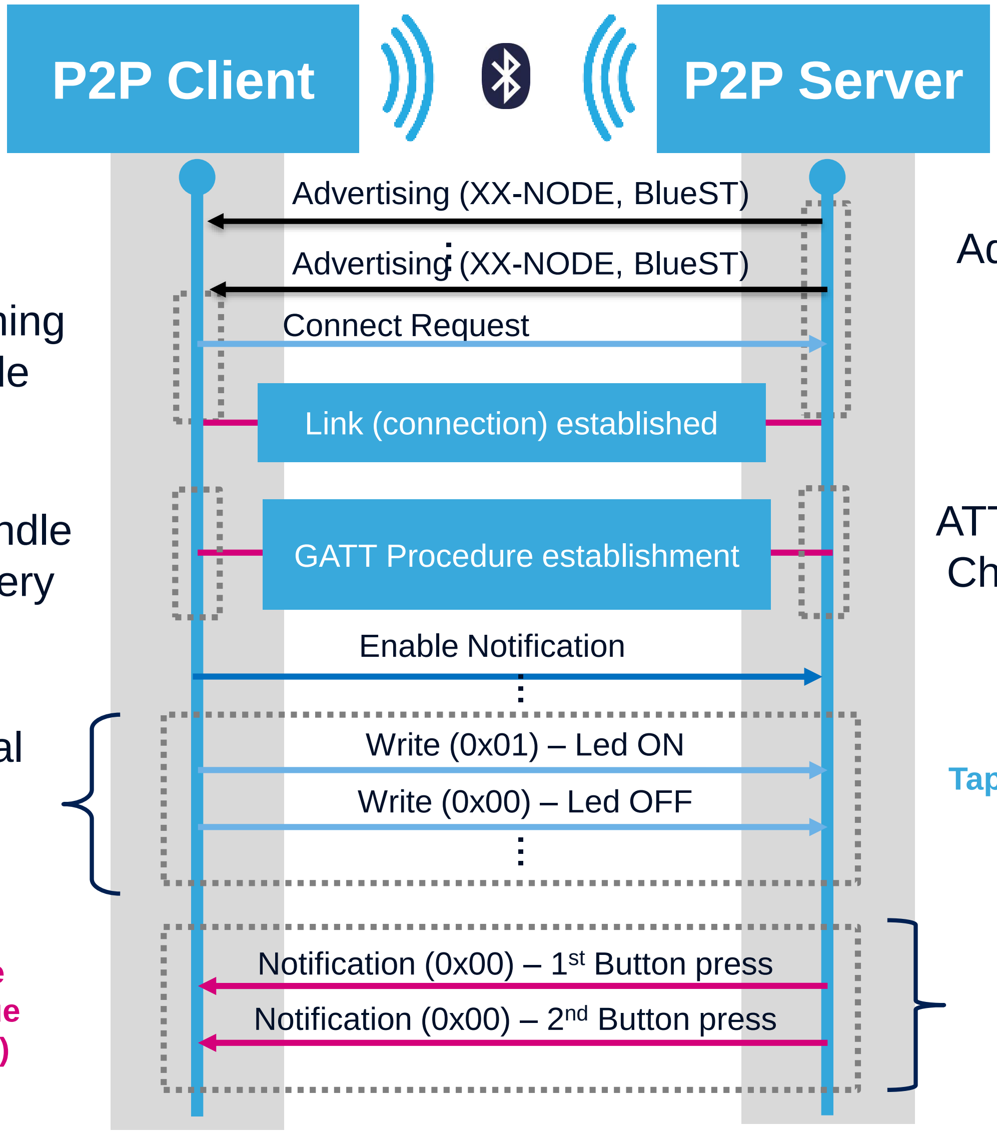
STM32 WB

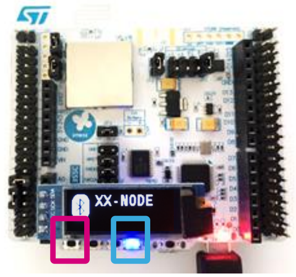
Finished !

Central **GATT Client**

Peripheral **GATT Server**




P2P Client
P2P Server
Advertising (XX-NODE, BlueST)
Advertising (XX-NODE, BlueST)
Connect Request
Link (connection) established
GATT Procedure establishment
Enable Notification
Write (0x01) – Led ON
Write (0x00) – Led OFF
Notification (0x00) – 1st Button press
Notification (0x00) – 2nd Button press



Scanning Mode

ATT Handle Discovery

Central-to-Peripheral communication **(Control SLAVE)**

Push SW1 to update characteristic #1 value (notify value change)

Advertising Mode

ATT Services & Characteristics

Tap Button in the phone app to write new characteristic #2 value

Peripheral-to-Central communication **(Notify MASTER)**

EXTRA

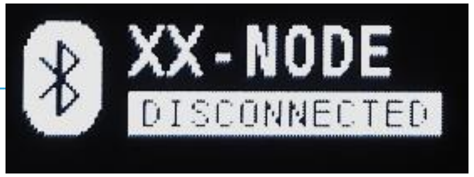
OLED display info extension

Display connection state

app_ble.c

SVCCTL_App_Notification(…) user section **{ EVT_DISCONN_COMPLETE }**

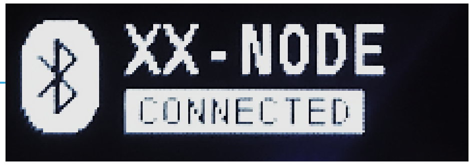
```
/* USER CODE BEGIN EVT_DISCONN_COMPLETE */
LCD_BLE_PrintStatus("DISCONNECTED");

/* Start the advertising again */
Adv_Request();
/* USER CODE END EVT_DISCONN_COMPLETE */
```



Restart advertising

user section **{ EVT_LE_CONN_COMPLETE }**

```
/* USER CODE BEGIN HCI_EVT_LE_CONN_COMPLETE */
LCD_BLE_PrintStatus("CONNECTED");
/* USER CODE END HCI_EVT_LE_CONN_COMPLETE */
```



Link lost (Disconnected) → *EVT_DISCONN_COMPLETE*
Link established (Connected) → *EVT_LE_CONN_COMPLETE*

life.augmented

STM32 WB

EXTRA

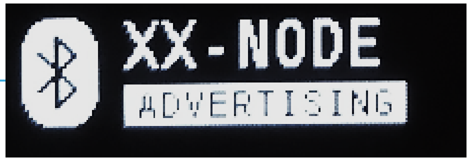
Add the OLED display info extension

Display advertising state

app_ble.c

Adv_Req() user section **{ Adv_Request_START_SUCCESS }**

```
/* Update Advertising data */
ret = aci_gap_update_adv_data(sizeof(manuf_data), (uint8_t*) manuf_data);

if (ret == BLE_STATUS_SUCCESS)
{
    ⋮

    /* USER CODE BEGIN Adv_Request_START_SUCCESS */
    LCD_BLE_PrintStatus("ADVERTISING");
```



EXTRA

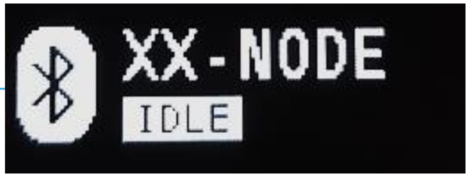
Add the OLED display info extension

Display idle state

app_ble.c

Adv_Cancel() user section **{ Adv_Cancel_START_SUCCESS }**

```
/*  Stop advertising */
ret = aci_gap_set_non_discoverable();

BleApplicationContext.Device_Connection_Status = APP_BLE_IDLE;
if (ret == BLE_STATUS_SUCCESS)
{


  /* USER CODE BEGIN Adv_Cancel_STOP_SUCCESS */
  LCD_BLE_PrintStatus("IDLE");
  /* USER CODE END Adv_Cancel_STOP_SUCCESS */
```



life.augmented

STM32 WB

EXTRA

Notification control extension

p2p_server_app.c

```
void P2PS_STM_App_Notification(P2PS_STM_App_Notification_evt_t *pNotification)
  {
  switch(pNotification->P2P_Evt_Opcode)
  {
    case P2PS_STM__NOTIFY_ENABLED_EVT:
      /* Client registered for notifications */
      ⁞
      break;
    case P2PS_STM_NOTIFY_DISABLED_EVT:
      /* Client unregistered for notifications */
      ⁞
      break;
    case P2PS_STM_WRITE_EVT:
      ⁞
      break;
    default:
    break;
  }
  return;
}
```

EVT_BLUE_GATT_ATTRIBUTE_MODIFIED GATT events propagated from ***PeerToPeer_Event_Handler(…)*** registered @SVCCTL during ***P2PS_STM_Init()***

P2P_NOTIFY Client Characteristic Configuration descriptor value changed

Notifications to be sent only if enabled from Client side

Attribute modified by client $\longrightarrow$ *EVT_BLUE_GATT_ATTRIBUTE_MODIFIED*

GAP and GATT commands and events used so far

GAP

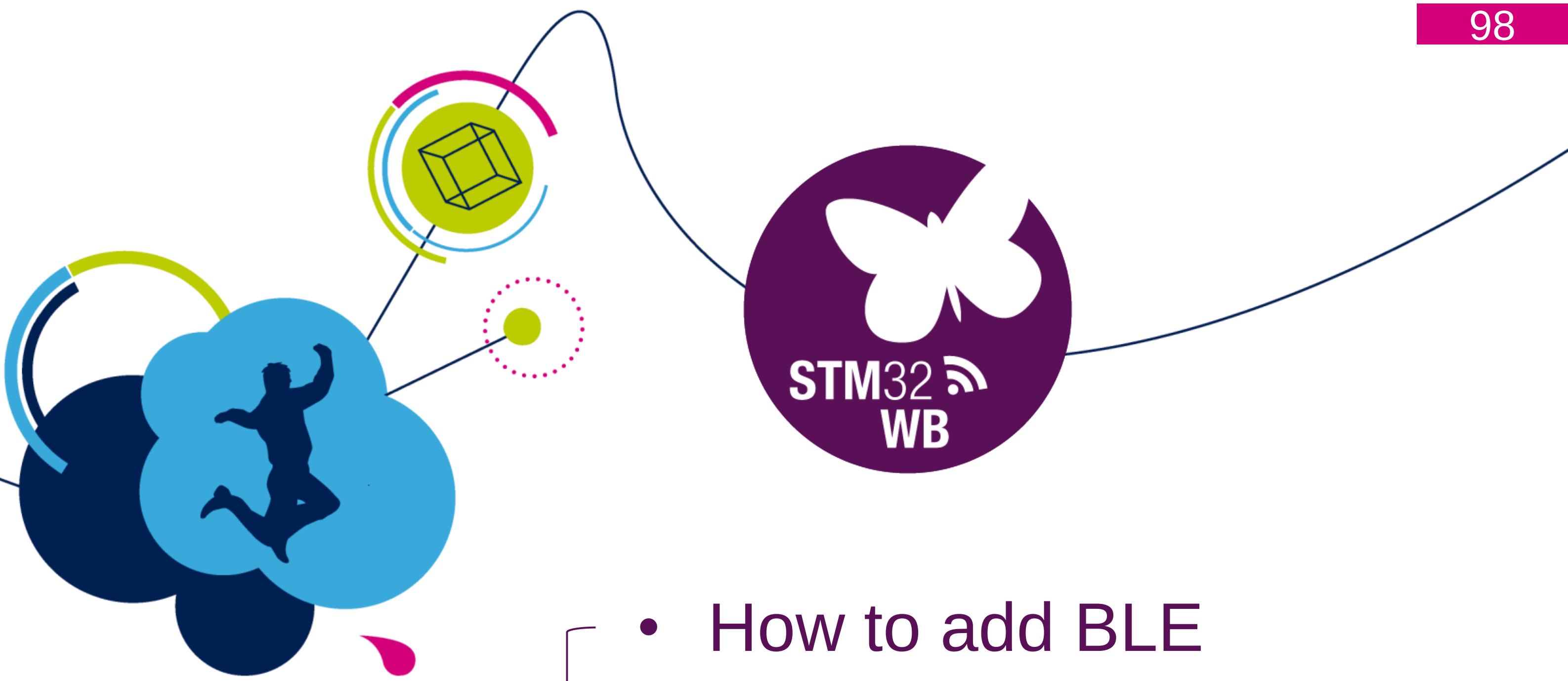
Start advertising →	*aci_gap_set_discoverable(…);*
Update advertising data →	*aci_gap_update_adv_data(…);*
Stop advertising →	*aci_gap_set_non_discoverable(…);*
Link lost (Disconnected) →	*EVT_DISCONN_COMPLETE*
Link established (Connected) →	*EVT_LE_META_EVENT (EVT_LE_CONN_COMPLETE)*

GATT

Add Service →	*aci_gatt_add_service(…);*
Add Characteristic →	*aci_gatt_add_char(…);*
Update Characteristic value →	*aci_gatt_update_char_value(…);*
Attribute modified by client →	*EVT_BLUE_GATT_ATTRIBUTE_MODIFIED*

life.augmented

STM32 WB



What we have learned?

- How to add BLE
- STM32_WPAN BLE basics
- P2PServer application
- BLE principles/terminology